NO. 17 MARCH 1984 $3.00

THE MAGAZINE FOR ATARI® COMPUTER OWNERS

A.N.A.L.O.G. COMPUTING

SPECIAL DISK ISSUE!

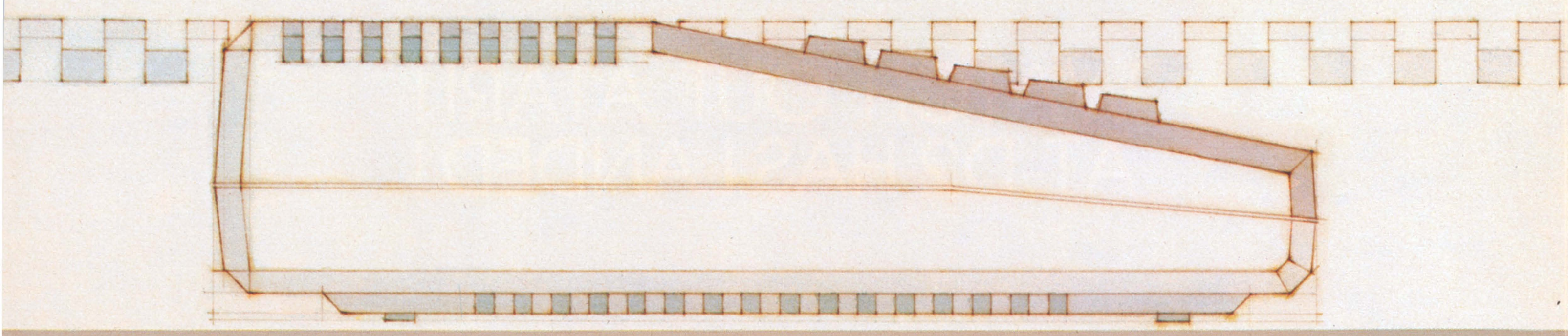
IN THIS ISSUE:

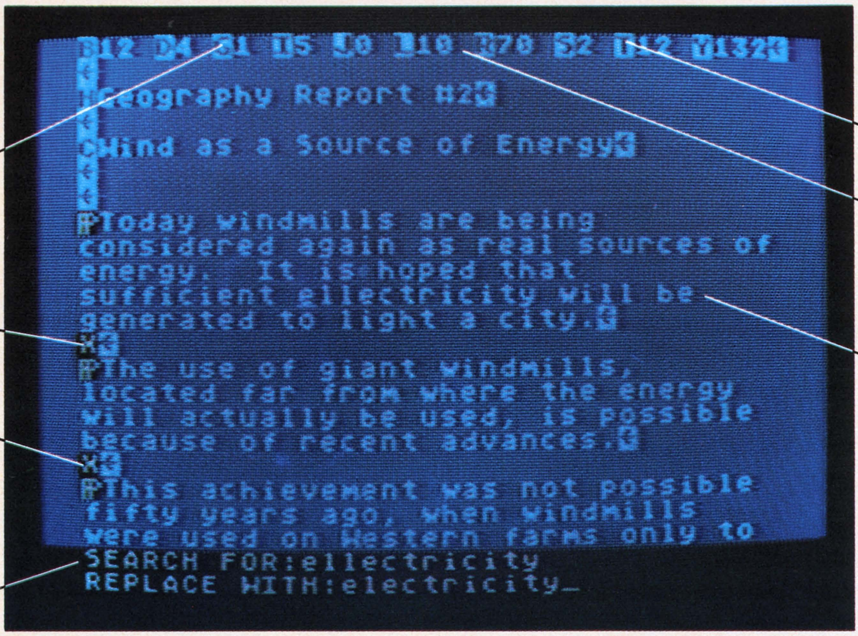
PLANETARY DEFENSE

SUPER MACHINE LANGUAGE GAME

0 74470 12385 03


WATCH OUT ATARI!
AT-D2 HAS LANDED!
trak
DENSITY S D
WRITE PROTECT
READ WRITE
TRACK NO
88





Prints with built-in format or lets you create your own: center, underline. Boldface, elongated, proportional and condensed print.

Inserts and deletes characters, lines or blocks of text.

Duplicates and moves blocks of text.

Searches for and/or replaces words or phrases, one at a time or all at once.

Changes line spacing, margins, print type, paragraph indents anywhere in the document.

Instantly reformats.

Enters text easily, with word wrap, one main menu and helpful prompts.

Creates multi-line headers and footers.

Prints double columns, form letters, multiple copies of a page or document, at the touch of a key.

Numbers pages and creates section numbers automatically.

Corrects spelling (with extra software).

Chains and merges files.

Has "print preview": you can check your copy before it's printed.

You can't find a friendlier, more powerful word processor at twice the price. New AtariWriter.™ Under $100.

Now you can do multi-featured word processing at home, simply. At a family budget price.

Our ROM-based cartridge technology means you can use new AtariWriter on any ATARI® Home Computer (even 16K) for personal and business correspondence, term papers, committee reports, mailings, etc.

It also lets you choose between cassette and disk storage systems.

One very special AtariWriter feature: you can correct as you write, without switching back and forth between Create and Edit modes.

And our memory buffer offers an "undo" command to let you change your mind, and restore text you've just deleted.

Check into our remarkable AtariWriter, and our choice of letter quality and dot matrix printers, also reasonably priced, at Atari dealers. Call 800-538-8543 for dealer nearest you. In California, call 800-672-1404.

You'll do more with Atari Home Computers.

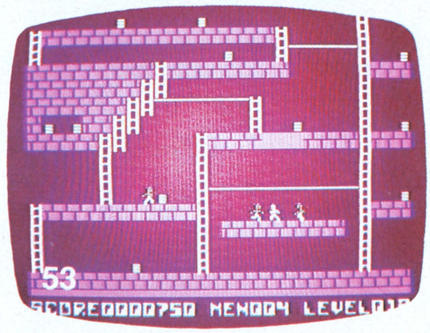
NO. 17 MARCH 1984

THE MAGAZINE FOR ATARI® COMPUTER OWNERS

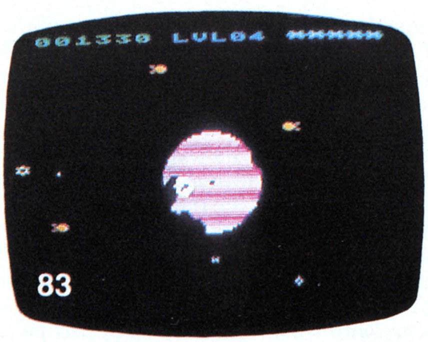
A.N.A.L.O.G. COMPUTING™

FEATURES

PRODUCT REVIEWS



COLUMNS



ANALOG COMPUTING (ISSN 0744-9917) is published monthly for $28 per year by ANALOG Magazine Corp., 565 Main Street, Cherry Valley, MA 01611. Second-class postage paid at Worcester, MA and additional mailing offices.

For subscription information and service call toll-free:

1-800-345-8112

in Pennsylvania call 1-800-662-2444

ADVERTISING SALES

ANALOG Computing
Home Office
Michael DesChenes
National Advertising
(617) 892-9230

Address all advertising materials to:
Michael DesChenes — Advertising Production
ANALOG Computing
565 Main Street, Cherry Valley, MA 01611

ANALOG COMPUTING (ISSN 0744-9917) is published monthly for $28 per year by **ANALOG** Magazine Corp., 565 Main Street, Cherry Valley, MA 01611, Tel. (617) 892-3488. Second-class postage paid at Worcester, MA and additional mailing offices. POSTMASTER: Send address changes to **ANALOG COMPUTING**, P.O. Box 615, Holmes, PA 19043. No portion of this magazine may be reproduced in any form without written permission of the publisher. Program listings should be provided in printed form. Articles should be furnished as typed copy in upper and lower case with double spacing. By submitting articles to **ANALOG COMPUTING**, authors acknowledge that such materials, upon acceptance for publication, become the exclusive property of **ANALOG**. If not accepted for publication, the articles and/or programs will remain the property of the author. If submissions are to be returned, please supply self-addressed, stamped envelope. U.S.A. Newstand distribution by Eastern News Distributors, Inc., 111 Eighth Ave., New York, NY 10011.

EDITORIAL

by Brian Moriarty

I have been an Atari 800 owner and subscriber to **ANALOG** *for about a year. Most of this time, there have been stories and rumors about Atari getting out of the home computer business. I have continued to invest time and money in my Atari, and hoping for the best.*

In the last two months, computer stores have been dropping the Atari line, and some of your advertisers have been returning my checks, saying that they no longer carry Atari software (e.g.; Eastcoast Software).

I would appreciate an explanation of what is happening at Warner/Atari, and your opinion on the future of Atari home computers. The natural tendency is to ignore a problem or sugar-coat it, but the most responsible action is to deal with it directly.

Clyde Lawrence
Miami, Florida

Nobody wants to know what's going on at Atari more than we do! Despite our status as a large and ancient Atari magazine, we have considerably less access to Warner's board chamber than *The Wall Street Journal.* But since you and many other **ANALOG** readers are asking, here's my limited view of what's going down in Sunnyvale.

Early last year, Atari lost its Prime Mover status in the low-end computer market to an undeserving Commodore. They also lost millions of dollars, hundreds of employees and corporate President Ray Kassar. Most importantly, they lost the confidence of their stockholders and retailers, and the buying public.

A lesser company would have shriveled up and blown away under this kind of pressure. It happened to Osborne; but Warner was determined that it was not going to happen to Atari. So they hired James Morgan away from Phillip-Morris (the cigarette conglomerate) to steer their failing Atari division away from the path to extinction.

So far, it looks as if Warner made a wise choice. Mr. Morgan is handling the fragile remains of Atari with caution and ruthless conservatism. He's slashing overhead, eliminating waste and concentrating resources in a heroic effort to keep his rapidly shrinking company afloat.

Some of Mr. Morgan's decisions haven't made him very popular. Many projects initiated under Kassar's flamboyant administration, including the 1400 and 1600XL computers and the CP/M Module, have been quietly cancelled. Major new product announcements are at a virtual standstill. Retailers are furious at Morgan's hard-line policies regarding product returns and quotas. But Warner didn't hire Morgan to play Mr. Popularity. They hired him to put Atari back in the black, and he is doing it in spite of the head-shakers on Wall Street and the "I told you video games were just a fad" attitude of the press.

Evidence of Atari's recovery isn't hard to find. 600XL and 800XL computers are selling as fast as Taiwan can build them. New Atari accessories and software titles arrive at our office on a weekly basis — and Morgan is still talking about releasing a 1450XLD in one form or another later this year, although I get the impression that he's doing it mainly for the sake of having something to show at CES. So, barring sudden corporate buy-outs, unexpected calamities or an Act of IBM, Atari's Home Computer Division will probably stay reasonably healthy for at least another year, perhaps much longer.

Now let me tell you why I had my fingers crossed. The home computer industry thrives on things that are New, Different and Exciting, adjectives which do not apply to the current lineup of XL machines. Atari is still the most advanced computer you can buy for the money. But how much longer will it be before somebody comes along and offers more for less? Rather than waste precious time on yet another 8-bit XL, I'd like to see Atari throw everything they've got into a second-generation machine that's as far ahead of the competition as the 800 was back in 1979. I know for a fact that Atari research has developed all sorts of wonderful new toys. How long will it be before Morgan will dare to unleash them? Summer CES isn't far away — and it may be Atari's last chance to maintain their position as a technological leader. □

CORRECTION FROM ISSUE #15

In the 600XL photograph on page 36, the labels on the ANTIC and GTIA chips were accidentally switched.

READER COMMENT

I am writing this letter to make a few comments about an article reviewing the new Atari 600XL in Issue #15.

I would first like to say that I enjoy your magazine very much, but this one particular article disturbed me.

First of all, it states that the 1400, 1450 and 1600XL projects have been put on "indefinite hold." Naturally I was worried, so I made a call to Atari itself and they confirmed that they were not on indefinite hold, but that their release dates were just rescheduled for later in the year.

Next, you said that the keyboard was almost comparable to the 800's, but I've seen a 600 and I own an 800, and I thought the 600's keyboard was significantly better.

To go on, you said that the 600XL "isn't as exciting or advanced as the 400/800 series when it first appeared in 1979." I say you're living in the past. Wake up! The expansion capability far surpasses that of any other computer on the market today. You also said, "These new XL computers are merely repackaged 800s." Not true! The new 600XL may not be different in some respects, but there are obvious changes for what I would call the better. They made the RAM expandable to 64K (not 48K as implied in your article). Having an on-board BASIC is also helpful to relieve you of the taking in and out of cartridges all the time.

One last note about Alan Alda. He is paid to do commercials as Alan Alda, not, as you say, "a box of Hawkeyes."

Andrew Urban
Ulysses, Kansas

The 1400 and 1600XL projects are no longer on "indefinite hold." According to Atari officials at January CES, they've been **cancelled**. *Whoever you talked to at Atari wasn't up to speed. The 1450XLD is still alive as of this writing, but Atari won't speculate on when it's coming or what it will be like when it does arrive. Still sounds pretty indefinite to me.*

You're entitled to your opinion on the relative merits of the 800 and 600XL's keyboards. As I stated in the article, the 600 keyboard is very good, especially considering the price of the unit. But everyone I've talked to who's used the 600XL for any length of time agrees that the old 800 keyboard is more comfortable.

The new parallel expansion bus does indeed make the 600 potentially more expandable than a 400 or 800, and I made a point of noting that fact in my article. But to say that the 600XL's expansion capabilities "far surpass any other computer on the market today" is ludicrous. Have you ever looked at the back panel of an IBM PC, or inside an Apple IIe? Those slots aren't designed to hold credit cards. Even the lowly Commodore 64 has significant expansion potential.

I did not "imply" that Atari made the 600XL's RAM expandable to only 48K. I said it outright! There may be 64K's-worth of RAM chips in their (so far nonexistent) memory expander, but only 48K is accessible to the user unless you disable the operating system, which leaves you with a brainless shell of plastic instead of a computer. The same goes for the "64K" 800XL and 1200XL machines. I refuse to pay lip service to Atari's and Commodore's misleading claims for the memory capacity of their products. By the way, the 800 is also expandable to "64K," using Mosaic's adapter board.

Your letter makes it sound as if I didn't say anything about the improvements found in the new XL line, such as the on-board BASIC and the HELP key. Reread my article carefully and you'll find ample praise for these and other new features. But these little bonuses are just a fresh coat of paint over a design that has remained essentially unchanged since 1979.

Let me conclude by denying that I ever said Atari was paying Mr. Alda to represent anyone but himself in his commercials. You've taken my "box of Hawkeyes" comment completely out of context. Just to be sure, though, I watched a couple of the commercials over again, just to be sure Alda wasn't wearing fatigues or a stethescope. He isn't. The ads are easy to find. Atari runs them during each weekly episode of **After M*A*S*H.**

—B.M.

I am having problems running your programs **Fill 'Er Up!** and **Livewire!** on my Atari 600XL.

I had a 400 that I made the self-booting tapes with, and they worked fine. But when I load the programs into my 600, it returns to BASIC at the end of the load. I would appreciate any hints you could give me to help with this problem.

I'm very happy to see **ANALOG** going monthly. And I think it's the best magazine for the Atari on the market.

John Merlino
Sherburne, NY

Try holding down both the START and OPTION keys together when you turn your computer on. The

START key tells your XL that you want to boot a tape; the OPTION key says that you want to disable the built-in BASIC. Disk users need only hold down the OPTION key during power-up to disable BASIC. Future **ANALOG** *games will include instructions to this effect, starting with this month's* **Planetary Defense**.
—B.M.

Roundup Joystick Button Restart

We have been enjoying Richard Loken's **Roundup** from Issue #13. There is one improvement we would very much like to add. Can you tell us how to do it?

When you have been playing the game from your easy chair (with a 12-foot cord on your joystick), it is very annoying to have to get up and go back to the console to push START for a new game. We know how to change program lines so that you can push the TRIGGER to restart a BASIC program. What do you change in a machine language game like **Roundup**?

We wish all game programmers would make a note of this: If your game does not require keyboard input, then don't make the user go back to the console solely to restart the program.

Carolyn Hoglin
Orlando, Florida

The following changes to the **Roundup** *boot-maker program will allow you to restart the game with the joystick trigger:*

```
32 IF PASS=0 THEN 40
34 IF LINE<>1520 OR X<>27
THEN 38
35 PUT #1,234:PUT #1,234:P
UT #1,173:PUT #1,132:PUT #
1,2:PUT #1,240:PUT #1,251:
PUT #1,173:PUT #1,132:PUT
#1,2
36 PUT #1,208:PUT #1,251:X
=49:? "PATCHING":GOTO 39
38 PUT #1,BYTE
39 NEXT X:READ CHKSUM:GOTO
 25
```

•

—T.H.

Disassembler Update

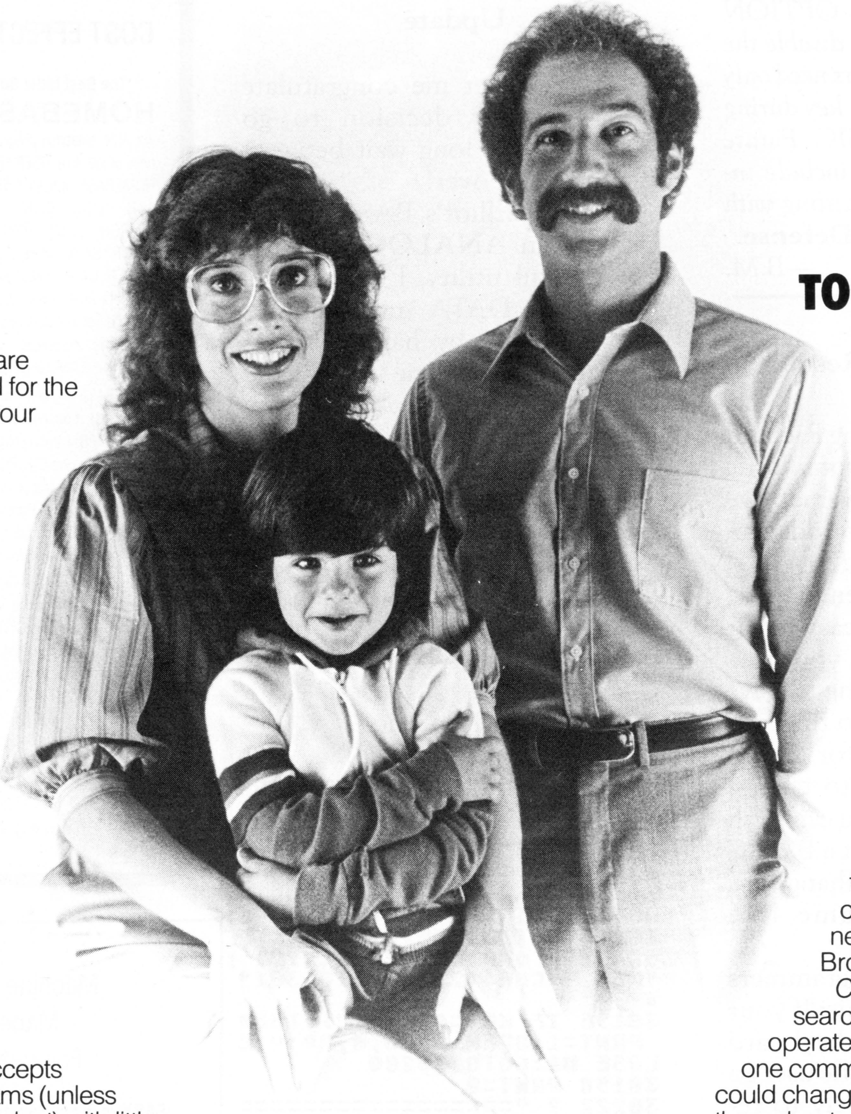
First off, let me congratulate you on your decision to go monthly. The long wait between issues is now over!

Maurice Elliot's BASIC disassembler in **ANALOG #14** is a very useful utility. I was always converting DATA statements to machine code by hand. Now, I have much more time to study the code, instead of looking it up. However, the program only allows output to the screen, which makes reading a little tedious. The few lines that follow, when added to the program, will produce a hard copy upon request. Studying the code now becomes, as Mr. Elliot put it, "a relaxing, armchair task."

Ed Schembri
Mississauga, Ontario

```
30015 OPEN #1,4,0,"K:":POK
E 710,0:POKE 712,0:POKE 75
2,1
30017 ? CHR$(125):? "Pleas
e wait......"
30192 ? :? :? "  DO YOU W
ANT A PRINTOUT? (Y/N)"
30194 GET #1,KEY:IF KEY<>A
SC("N") AND KEY<>ASC("Y")
THEN ? CHR$(253):GOTO 3019
4
30196 IF KEY=ASC("Y") THEN
 PRNT=1:OPEN #4,8,0,"P:":C
LOSE #1:GOTO 30200
30198 PRNT=0
30222 ? "=================
================"
30224 IF PRNT THEN ? #4;"A
DDR DEC    OPN   OPERAND
   DEC"
30226 IF PRNT THEN ? #4;"=
==========================
======"
30295 IF PRNT THEN ? #4;PC
$,OPCODES$(3*OPNUM-2,3*OPN
UM),
30525 IF PRNT THEN ? #4
30625 IF PRNT THEN ? #4;"#
$";HEX$,,OPRND
30745 IF PRNT THEN ? #4;"$
";HEX$
30825 IF PRNT THEN ? #4;"$
";HEX$,,OPRND
30925 IF PRNT THEN ? #4;"$
";HEX$;",X",OPRND
31025 IF PRNT THEN ? #4;"$
";HEX$;",Y",OPRND
31125 IF PRNT THEN ? #4;"$
";HEX$,OPRND
31225 IF PRNT THEN ? #4;"$
";HEX$;",X",OPRND
31325 IF PRNT THEN ? #4;"$
";HEX$;",Y",OPRND
31425 IF PRNT THEN ? #4;"(
$";HEX$;")",OPRND
31525 IF PRNT THEN ? #4;"(
$";HEX$;",X)",OPRND
31625 IF PRNT THEN ? #4;"(
$";HEX$;",Y)",OPRND
```

ATARI SOFTWARE FOR THE WHOLE FAMILY



Here are four software packages designed for the different people in your family.

A BASIC COMPILER FOR THE PROGRAMMER

ABC (A BASIC Compiler) automatically translates Atari BASIC programs into high-performance integer P-code that runs up to 12 times faster!

Perfect for developing system software and commercial games, ABC accepts most BASIC programs (unless floating point dependent) with little or no modification. Compiled P-code is a self-standing DOS object module that is unLISTable and runs without the BASIC cartridge.

ABC allows expressions in DIM, GOTO, GOSUB, and RESTORE statements, doesn't require you to re-order lines, and fully supports string and sub-string operations.

Give your BASIC programs the look and "feel" of professional products with ABC. 40K Disk ***$69.95***. Manual alone ***$9.95*** (credited toward compiler purchase).

MAKEBOOT lets you create self-booting disk or cassette versions of your ABC compiled software. Reduces overall program load time and saves memory and disk space by eliminating DOS. 40K Disk ***$14.95***.

AN EDUCATIONAL TOY FOR PRE-SCHOOLERS

Monarch is proud to present ***SofToy***, an educational program smart enough to act simple.

Bells ring, balls bounce, owls hoot as SofToy and its colorful interactive display gently introduce children (two years and older) to spatial relations, letters, numbers, even elementary programming! SofToy lets kids become familiar with computers, without arbitrary demands, competition, or intimidation. SofToy grows with children, too. At more difficult levels, the match game is a real challenge for the whole family. 24K Disk ***$29.95***.

TOOLS FOR THE SERIOUS USER

Power Tools I combines four sophisticated text processing tools on one easy-to-use utility disk.

DIFF shows you differences between two ATASCII text files: for example, changes you made in a program or document from one version to the next.

Manually searching for a particular text file can take hours. But now, with *SEARCH,* you specify a search string and a list of files. *SEARCH* examines the files and points out which ones have that string. You'll never lose your Fudge Brownie recipe again!

CHANGE is a powerful search and replace utility that operates on multiple files with one command. For example, you could change character names throughout your novel with a single command, even if each chapter is a separate file.

The special pattern-matching and multiple disk capabilities of *SEARCH* and *CHANGE* are an added plus.

TRANSLIT lets you swap one character set for another (for example, upper case for lower case) throughout a file with one command.

Power Tools I is ideal for professional business and software development text applications. 40K Disk ***$34.95***.

Look for these fine products at your nearest ATARI dealer, or order direct. Include $2.00 for shipping and handling in U.S./Canada. Checks or money orders accepted. MasterCard/Visa add 3%. COD higher. MA residents add 5% tax. Dealer and distributor inquiries invited. ATARI is a trademark of Atari, Inc. ABC, Makeboot, Power Tools, and SofToy are trademarks of Monarch Data Systems, Inc.

MONARCH DATA SYSTEMS

MONARCH DATA SYSTEMS, INC., P.O. Box 207, Cochituate, MA 01778 (617) 877-3457

In your article about the new 600XL computer, you asked for readers to let you know about problems. Well, here goes. I am a **teacher at the local high school**, and I run introductory computer courses for adults. We use Atari 800s and I recommend them to anyone who asks.

A friend of mine bought a 600XL as a Christmas present for her grandson, and has had incredible troubles. She purchased the program **States and Capitals** for the child's use. However, the program generates an ERROR 2 (out of memory). Repeated calls to the Atari hotline were fruitless; the line was busy all the time.

After several hours of experimentation, we tried what we thought was an off-the-wall idea; to insert an old BASIC cartridge into the slot to see if this made a difference. It did. The program was able to run with no problems! Apparently, the program addresses the memory of the 600XL differently than it would with the 400 or 800. This also happened with the Atari program **Scram**. I felt very embarrassed, having raved about the Atari as a superior computer over the Commodore and the TI.

One good note, though. The owner, in a conversation with another person from Atari, Inc., was told that she would be sent a BASIC cartridge at no cost, so that she would be able to use the programs. Nice touch from Atari, although it would have been nicer not to have gone through the grief.

Just a couple of thoughts, while I'm here at the keyboard. First, congratulations on having gone monthly. It is terrific to now read your ideas and programs more often during the year.

There is now a national group of educators who use the Atari computer. It is called the Atari Teachers Network. Dues are $4.00 a year. This type of organization is critical in helping to bolster Atari's growth in the education field. The address is:

Atari Teachers Network
P.O. Box 1176
Orange, New Jersey 07051

Finally, it is nice to see your new column on educational software. I think that it is long overdue in your magazine. If Dr. Griffin needs any help digging through the mountain of software he has, let him know that I and my students would be more than happy to field test and review some for him.

Rick Paula
Quabbin Regional High
Barre, MA

To The Editors:

Alexander Leavens' comments on software piracy in Issue 15 struck a chord or agreement in my heart. Nevertheless, no matter how much I might emotionally sympathize with his ideas, he offers no solutions for the problem. I too am a software author, perhaps not as successful as Alex, since my royalties fall far short of paying my rent. But the problem of piracy is one which affects us all, whether we write programs or just use published material.

Pirates affect the software industry as a whole the same way that any black market subverts the legitimate free enterprise system; they siphon off money which keeps the software business afloat. Like any parasite, they can kill their host. They might just do that, since the video games industry is far from being in its prime anymore.

But poor Alex rants and raves against the thieves and, I suppose, hopes to intimidate them by calling them immoral bad guys, etc., etc. That won't stop software thieves, believe me.

In the Real World (which he often refers to), there is only one technique that will stop crime of any sort: make the risks exceed the likely benefits from the deed. So far, the only approach that has been tried to accomplish this is software security protection. It is normal for programmers, who rely on the marvels of modern technology for their daily bread, to cling to a technological cure for the cancer of software theft. Unfortunately, whatever can be secured by Man or machine can also be made insecure. Protection schemes, like all security gimmicks, only delay the theft of the precious products.

No security scheme can prevent piracy. The question is only how much time and effort (or fancy disk/cartridge disassembler equipment) the criminal is willing to invest in to beat the protection schemes. When you're talking potential profits of tens, even hundreds of thousands of dollars from purloined programs, rest assured that the thief is willing to invest quite a bit of work and money to get copies of a successful game. Appealing to his morality and sense of fair play as a deterrent is a joke.

Even though I have little faith in government jurisdiction over anything, the only possible resolution seems to be legal recourse. The threat of a nice long jail sentence

and a hefty fine could do the trick. Unfortunately, our legal system, as sorry as it is anyway, is completely and totally unprepared to deal with the issue of computer crime. Just read the legal briefs on the subject and you will see that most every successful pirate, when caught, is slapped on the wrist by the confused courts and thereafter becomes rich and famous from his or her exploits by writing a book or becoming a consultant. The ones that don't get caught remain secure in the knowledge that there are no realistic penalties for this type of crime.

Step number one is for authors and members of the software industry to lobby for sensible copyright legislation. The current copyright laws were OK for the 1800s, but are totally obsolete today. Here is where they fail:

a) Current laws do not go far enough to grant an "identity" to a software product. If a program cannot easily be distinguished from its copy, the law, in practice, does not consider them versions of the same original. In other words, if the pirate copies a proprietary product and makes a couple of minor changes to it, the courts often cannot identify the result as a copy and will not do so without a lot of expert testimony and expensive, drawn-out court proceedings to determine the truth. If copyright laws were modeled more after existing patent office standards, or if software were to be covered by patent laws, this problem and a few others would be resolved. I say let software be governed by patent laws, not by copyrights. Programs are "unique processes" and should fall under the patent umbrella.

b) The law does not have any means to pursue pirates. At present, pirates can go about their business undetected because there is no mechanism for finding out whether a program is an illegitimate copy or an original. A stolen program runs the same as one that was honestly purchased. I would like to see laws which set a uniform protection standard (serial number recognition, etc.) for software and which requires hardware (Atari, Commodore, etc.) manufacturers to include devices that can recognize the difference between legal and illegal software. Yes, I am calling for a little of Big Brother's influence here and, as distrustful as I am of technological solutions to human problems, I demand legally prerequisite standards for security hardware. The stakes are Big too, folks!

c) The penalties for copyright infringement should go beyond mere recompensation of lost income for the author/publisher, and set standards for severe punitive measures. Send Dirty Harry after the pirates.

d) Last and most important of all, software authors, publishers and hardware manufacturers should join forces and lobby for whatever legislation will be the most beneficial in curbing software piracy. These little letters to the editors are nice and give vent to our spleen, but they do nothing to solve the problem. Let's go to Washington!

Oh, and one more thing. The video recording industry is suffering from some of the same problems as ours. Let's keep a close eye on what those folks are doing, since they have the weight of the movie moguls and some very high-priced legal help behind them. The VCR folks can do a lot to break important legal ground and set precedents that we might want to follow. It might even be beneficial to join forces with the Big Guys. We must hang together or we shall surely all hang separately!

Reinhard Mirkovich
Billerica, MA

NEW PRODUCTS

by Lee Pappas

The Winter Consumer Electronics Show, held the first week of January in Las Vegas, is one of the industry's traditional new-product introduction periods. No major hardware announcements were made by Atari, and computer dealerships and Toys 'R' Us stores have yet to see most of the hardware exhibited last June in Chicago. Atari is apparently still in a regrouping stage after the takeover of James Morgan, the latest chairman and CEO.

Hardware: Now you see it...

600XL and **800XL** computers have been shipping in quantity for several weeks now, but don't give up your old 800 in anticipation of a **1400XL** or **1450XLD** just yet. The **1400** has been officially dropped from the product line. The **1450XLD** is still planned for release "sometime in the near future."

Another victim of Mr. Morgan's axe is the **CPM Module**. This could have been a breakthrough product for advanced Atari users; hopefully someone else will take the idea and put it on the market. One reason for the **CPM Module**'s demise may be that Microsoft's MS-DOS has taken all the steam out of CP/M in the influential IBM PC market.

Products touted last June that should see dealers shelves soon are the **Light Pen**, the **AtariLab** package, the **XL Expansion Unit** and the **Atari Touch Tablet**. The upcoming **1064 Memory Expander** will upgrade 600XLs to 64K (48K effective) of RAM.

Not just fun 'n games.

New Atari software titles include **AtariMusic I** and **II** and **Typo Attack**, an entertaining typing teacher originally marketed by the Atari Program Exchange. New "practical" programs include the **SynApps** series from Synapse Software: **SynTrend**, a two-stage statistics and graphics package; **SynCalc**, a VisiCalc-type spreadsheet; and **SynFile**, a database management system. All three require a 48K disk systems and retail for $99.95 apiece.

Registered owners of Atari's **1050 Disk Drive** can write to Atari and obtain a free copy of their new **DOS 3.0** high-density disk operating system, along with the publications *Introduction To DOS 3* and *DOS 3 Reference Manual*. The address is:

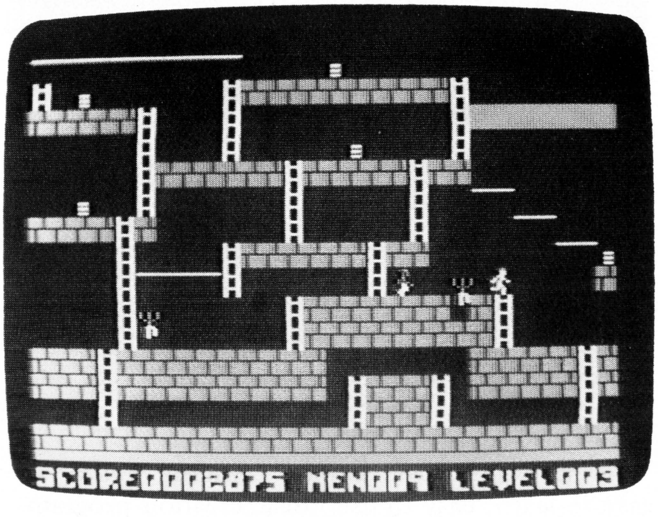
1050 Free DOS 3 Promotion
Atari Customer Relations
1312 Crossman Road
P.O. Box 61657
Sunnyvale, CA 94088

The games.

Mario Brothers is Atari's second **Donkey Kong** sequel, the first being **Donkey Kong Jr.** There are no gorillas in this game, just a lot of pipes and weird things blocking the pipes. **The Legacy** isn't too far-fetched. It takes place after a nuclear war; the survivors must discover and destroy retaliatory enemy silos. **Millipede** is a super-**Centipede** with bees, mosquitos, inchworms and other bugs. All retail for $49.95. Atari and Disney are working together to release **Captain Hook's Revenge**, a two-part disk based game ($44.95, 32K) designed with education in mind.

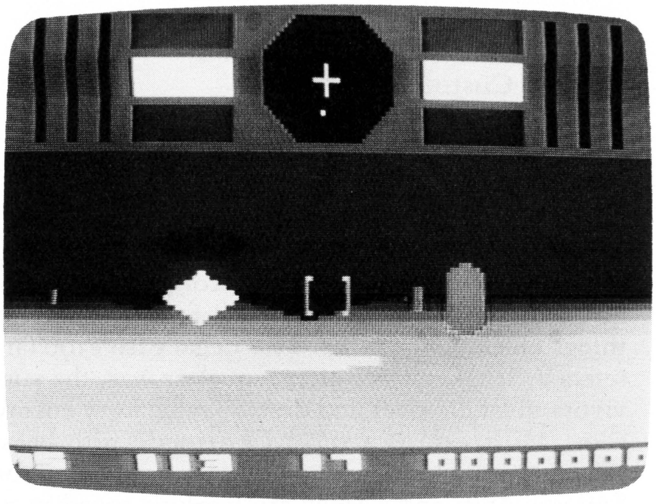
New from the third party.

A few days ago **ANALOG** received half a dozen new software products from as many companies. These consisted of a few less-than-terrific games: **Caverns of Callisto** from ORIGIN SYSTEMS, **Diamond Mine** from ROKLAN, and **Buck Rogers/Planet of Zoom** by SEGA. But my day was brightened with **Coloring II** by KOALA (for use with their **KoalaPad**), and **Operation Whirlwind** and **Lode Runner** from BRODERBUND.



Lode Runner.

SYNAPSE adds to its list of recreational software with **Encounter!**, a Battlezone lookalike. Hard on its heels is **Rainbow Walker**, a jump-on-the-grid game where you maneuver a small figure, Cedrick, on a "rainbow surface." The dreary grey segments turn to brilliant colors as you leap along avoiding holes, fragile squares, twitching devils and a multitude of other creatures I've never heard of. Interesting scrolling and a nerve-wracking bonus level will keep you hopping.

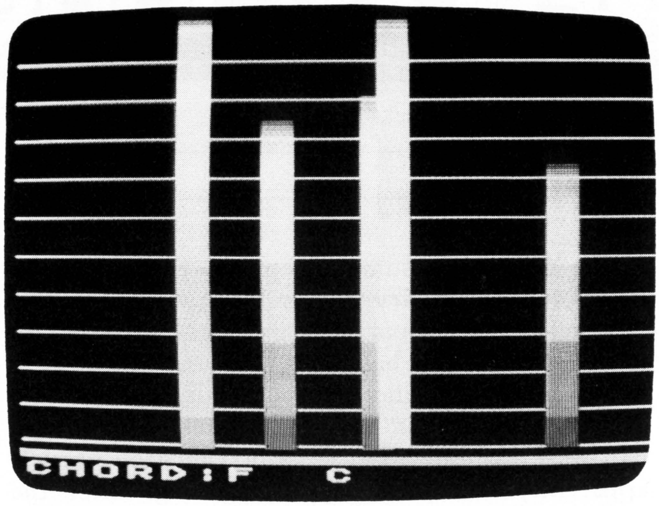


Encounter!

Rainbow Walker.

We await the release of Synapse's **Dimension X**, a dynamic graphics-oriented game; **Slamball**, a pinball game where *you* are the ball; **Reptilian**, a fast-action shoot 'em up; **Quasimodo**, in which you must keep clear of guards and recover jewels; **Alley-cat**, where Freddy the cat gets into all kinds of mischief; and **New York City**, an adventurous visit to the Big Apple.

From SOFTSYNC comes **Dancing Feats**, an easy-to-learn, easy-to-use music creation program. Multi-colored "bar graphs" dance to the beat of music which you have composed or typed in from sheet music. Also from Softsync comes **Personal Accountant**, **Computer Mechanic**, **Model Diet** and an arcade style game, **Mothership**.



Dancin' Feats.

HESWARE is releasing **Gridrunner**, **Attack of the Mutant Camels** (formally **Gridrunner II**), **Mr. TNT**, **Ghost Manor/Spike's Peak**, **The Pond**, **Missing Links**, **Factory** and **Rootin' Tootin'**; many of these are cartridge based. New educational software from CBS includes the **Success With Math Series**, which covers addition & subtraction, multiplication & division, linear equations, and quadratic equations.

TIGERSOFT's **Tiger Graphics** is a FORTH-written utility requiring 48K. It lets you put horizontal and vertical scrolling in your programs, along with custom display lists (including the GTIA modes) and redefined character sets. Other features include four-color character sets, mixing bit-map graphics with characters, and picture-drawing.

Other utilities in our morning mail include **Mega-Font** by XLENT SOFTWARE, which allows you to list programs containing special and inverse characters, print out text files, and get graphic dumps of mode 7+ and 8 screens. MMG, the publisher of **Final Flight!**, has released several helpful programs, including **RAM Test II** for $29.95, **Graphic Titler** for $39.95, **Payroll Package**, **Inventory Control**, **Data Manager**, **Accounts Receivable**, **Accounts Payable** and **General Ledger**. The business titles list for $99.95 and require 40K of RAM and at least one disk drive. All support a printer, and some require it. **Mail List** ($29.95) and **Form Letter Writer** ($29.95) round out MMG's application series. These latter programs can interface with most of MMG's business packages for mailing individual letters. Finally, MACROTRONICS is offering a new 32K program that can dump Atari graphics to most of the popular printers, including Epson, NEC 8023A, Trendcom 200 and Centronics 739. □

WHAT IS D:CHECK/C:CHECK?

Most program listings in **ANALOG** are followed by a table of numbers appearing as DATA statements, called "CHECKSUM DATA." These numbers are to be used in conjunction with D:CHECK and C:CHECK, which appeared in the **ANALOG Compendium** and Issue No. 16.

D:CHECK and C:CHECK are programs by Istvan Mohos and Tom Hudson. They are designed to find and correct typing errors when entering programs from the magazine. For those readers who do not have a copy of either article, send a pre-addressed, stamped, business-sized envelope to:

D:CHECK ARTICLE
P.O. BOX 23
WORCESTER, MA 01603

DRAPER PASCAL 1.5
by Norm Draper
DRAPER SOFTWARE
307 Forest Grove
Richardson, Texas 75080
48K Disk $79.95

by Brian Moriarty

Despite its wide following among educators and professionals, the Pascal language hasn't been very popular with Atari programmers. Why is this? I think the blame lies partly with Atari itself. Their semi-official **Pascal Language System** (APX-20102, $49.95) requires a minimum of two disk drives and the patience of a saint to operate — and, until now, it's been the *only* version of Pascal available. Small wonder that serious Atari users, many of whom presumably have had at least some prior exposure to Pascal, have turned to BASIC, FORTH and other languages rather than put up with the Atari dinosaur.

Draper Software's **Pascal 1.5** is a new and welcome addition to the growing list of Atari languages. Designed for use with only one or more disk drives, the Draper system offers Atari programmers a chance to flex their Pascal muscles without the thumb-twiddling and cursing that characterize its predecessor.

A Texan dialect.

Draper Pascal is not a full implementation of either the ISO or UCSD standard models of the language. In this respect, it falls short of **Atari Pascal**, which boasts full compliance with ISO. Draper's shortcomings include its abbreviated range of data types, limitations on parameter passing and program structuring, a lack of built-in number formatting and other, more subtle points too numerous to supply in detail.

Despite these eccentricities, **Draper Pascal** isn't likely to offend anyone familiar with more "polite" implementations of the language. It's close enough to the Real Thing to make it a perfectly legitimate tool for teaching and other formal applications.

Pascal is by design a generalized language, with almost no direct support for machine-specific concepts like PEEKing and POKEing. But Atari programmers would be hard-pressed indeed to get along without some kind of interface to the hardware. **Draper Pascal** addresses this need by offering lots of machine-specific features including I/O, disk management, string manipulation, BASIC-type graphics and sound, plus a handful of functions for reading game controllers and generating random numbers. Especially naughty by academic standards is Draper's CALL procedure, which allows you to easily access your own machine-language subroutines. **Listings 1** and **2** will show you what else the Draper vocabulary has to offer.

Standard data types recognized by **Draper Pascal** include 8-bit CHARs, 16-bit signed INTEGERs, STRINGs of up to 255 characters, REAL variables stored in the Atari's usual 6-byte floating point format, one-dimensional ARRAYs of any type (including STRINGs!) and a BOOLEAN flag for true/false testing. There's also a FILE type that is used by the system for handling various I/O chores. Subrange data types, RECORDS and pointers are not supported. Formal parameter passing is limited to INTEGER values, but you can always "pass" other types of values by using global variables. Likewise, functions can formally return only INTEGER values.

Listing 1.
Procedures, declarators and directives.

ARRAY	BEGIN	BLOAD
BOOLEAN	CALL	CASE
CHAR	CLOSE	COLOR
CONCAT	CONST	DELETE
DOS	DRAWTO	DUMPSTK
DVSTAT	EXIT	FILE
FOR	FUNCTION	GOTOXY
GRAPHICS	HIMEM	IF
INSERT	INTEGER	LOCK
MAXGRAPH	NOTE	OPEN
OPTIONS	PLOT	POINT
POKE	PROCEDURE	PROGRAM
PURGE	READLN	REAL
REPEAT	RESET	REWRITE
SETCOLOR	SOUND	STATUS
STRING	TRACEOFF	TRACEON
UNLOCK	VAR	WAIT
WHILE	WRITE	WRITELN
XCTL	XIO	

Listing 2.
Functions and operators.

ABS	ADDR	AND
ARCTAN	ASC	CHR
COPY	COS	CVTREAL
DEG	DIV	EOL
EOLN	EXP	EXP10
IORESULT	KEYPRESS	LENGTH
LN	LOCATE	LOG
LPENH	LPENV	MOD
NOT	ODD	OPTIONKEY
OR	ORD	PADDLE
PEEK	POS	PTRIG
RAD	RND	SELECTKEY
SHL	SHR	SIN
SQR	SQRT	STARTKEY
STICK	STR	STRIG
VAL	=	+
-	>	< >
<=	>=	

Pascal wouldn't be Pascal without a variety of program structures to choose from. Draper's system has most of the expected ones, including IF/THEN/ELSE, FOR/TO and FOR/DOWNTO, CASE with a nice ELSE option, REPEAT/UNTIL and WHILE. There is no GOTO in **Draper Pascal**, and you're probably better off without it. Both nested procedures and recursion are supported by the compiler. Forward references are not allowed.

Checking the oil.

Internally, the Draper system is fairly typical. There's an *editor* for composing source text, a *compiler* that translates the source into executable form, and a *supervisor* that provides run-time support for compiled programs. All three are supplied on a single copy-protected disk, along with 80 pages of looseleaf documentation and a registration form.

Draper's concisely written *User's Manual* assumes prior familiarity with Pascal. It comes with a sample run-through that shows you how to edit, compile and run your Pascal programs. A system glossary then explains each procedure and function in detail, with examples and Atari BASIC equivalents when applicable. You'll also find the complete Pascal source code for Draper's Editor and Main Menu routines, both in printed form and on the master disk. Very considerate.

The Draper editor is a modest but serviceable affair that lets you enter source text and edit it on a line-by-line basis. Single-character commands allow you to create new files, insert and delete individual lines or blocks of lines within a file, search for strings within a range of lines and save your files on disk for later editing and/or compilation. All lines must be referenced by an editor-supplied line number, although the numbers are not saved in the file or needed by the compiler.

The editor's buffer will hold a maximum of 250 eighty-character lines. This might limit the size of your programs were it not for an INCLUDE directive that supports unlimited chaining of disk files for compilation. The editor is also capable of saving and loading files from cassette. Too bad the compiler only accepts disk files as input.

Draper's line editor is a little too primitive for my taste. Those line numbers are a nuisance, especially when you don't have the full-screen editing of Atari BASIC to fall back on. I find it easier to use a general-purpose word processor like **Text Wizard** or **AtariWriter** to create my Pascal source files, saving Draper's line editor for interactive touch-ups and debugging. Just keep your lines less than eighty characters long, and the compiler will never know the difference. Note that **Atari Pascal** doesn't come with any editor at all; they expect you to buy their surrealistic **Program-Text Editor** (APX-20075, $39.95). In this context, Draper's line editor can be viewed as a handy little freebie.

P-coding.

Finished typing in that Pascal masterpiece? Then it's time to look at Draper's compiler, which converts your hours of labor into a working program you can be proud of.

The **Draper Pascal** system employs a single pass pseudo-compiler. That doesn't mean it's a fake; the term is used to distinguish so-called true or native-mode compilers which generate actual machine code from compilers like Draper's which generate pseudo-code or p-code. P-code can be regarded as the "machine language" of Draper's Run-Time Supervisor, a machine-language program that turns your Atari into a virtual Pascal machine. What a mouthful!

You must insert your original **Draper Pascal** disk in drive #1 before calling the compiler, or the copy protection won't let you do anything. Sigh. Once inside, all you have to do is specify the name of the Pascal source file and the name of an output file for the p-code. Make sure a copy of the Draper compiler is in drive #1, and you're off.

You can monitor the progress of the compile on your screen or printer. Each line of text is displayed, along with a hex offset value representing the address of that line relative to the beginning of the program. This information can be used together with Draper's TRACE option for debugging purposes. If a compile error is detected, you get an error message number, a brief diagnosis of the problem and a little arrow that indicates exactly where the compiler thinks something went wrong. Error-free compiles are rewarded with a table showing how many instructions and table entries your program is using, and the relative offset address of each variable.

The design of the Draper compiler places a couple of potential limits on the maximum size of your program. First, both source and output files must reside on the same physical disk. This is no big problem if you've got two drives, especially if they happen to be double density (an option which is fully supported by the system's custom DOS). Complications arise if you have just one drive. Because a copy of the compiler must be present on drive #1 in order to operate, your source and output files must also be able to fit on that same disk. I have yet to write a program that came even remotely close to filling a single-density disk. Nevertheless, a compiler that was completely memory-resident would certainly make things simpler.

Second, the compiler limits you to a maximum of 250 "table entries" per program. One entry is required for each variable declaration, each procedure and function name and each procedure or function parameter. As mentioned above, the compiler tells you how many entries you've used at the end of a compile. It's hard to say how complex an application would have to get before you'd start

running out of table space. My experiments have never used more than a handful of entries; but I haven't tried anything ambitious, either.

Fortunately, the system includes a very nice facility for chaining programs. Draper's XCTL (Transfer Control) procedure lets you automatically load and execute any compiled Pascal file under program control. If the specified file can't be found on the disk, the Supervisor will step in and ask for the proper disk. This feature may be just the thing for programs that are too big to be compiled in one big chunk.

Another strong point is the ease with which you can create practical, ready-to-boot applications. Simply rename your compiled file "INIT.PCD" and copy it over to a fresh DOS disk along with a copy of the AUTORUN.SYS file containing the Draper Supervisor. You'll end up with a genuine "turnkey" package that loads and executes without any user intervention. Draper offers a free 5-year run-time license to programmers who wish to sell applications written with their system.

Performance.

I was curious to see how **Draper Pascal** stacked up to other Atari languages in terms of speed and efficiency, particularly **Atari Pascal**. So I trotted out my trusty benchmarks **Sieve** and **Screen-Fill** (still warm from last month's **Action!** review — refer to that article for details), and threw them up against both the Draper and Atari compilers.

```
(* Sieve benchmark in Draper Pascal *)

PROGRAM SIEVE;

VAR COUNT,I,K,PRIME,TIME: INTEGER;
    FLAGS: ARRAY[8190] OF CHAR;

BEGIN

  POKE(559,0); (* ANTIC off *)

  POKE(19,0);  (* zero system timers *)
  POKE(20,0);

(* init flag array an( COUNT *)

  COUNT:=0;
  FOR I:=0 TO 8190 DO
    FLAGS(I):=1;

(* sieve algorithm *)

  FOR I:=0 TO 8190 DO
    BEGIN
      IF FLAGS(I) THEN
        BEGIN
          PRIME:=I+I+3;
          K:=I+PRIME;
          WHILE K<=8190 DO
            BEGIN
              FLAGS(K):=0;
              K:=K+PRIME
            END;
          COUNT:=COUNT+1
        END;
    END;

(* fetch timer reading *)

  TIME:=PEEK(20)+256*PEEK(19);

  POKE(559,34); (* restore screen *)
  WRITELN(COUNT,' PRIMES IN');
  WRITELN(TIME,' JIFFIES')

END.
```

Listing 3.

Listing 3 is my **Draper Pascal** implementation of the **Sieve**. It requires 6551 jiffies, or about 109 seconds, to execute on my 48K 800 system. This is roughly three times as fast as Atari BASIC at 19,490 jiffies. **Atari Pascal** ran the same algorithm in only 999 jiffies, more than six times faster than Draper. Interesting.

Next comes **Listing 4**, a Draper implementation of **Screen-Fill**. I obtained readings of 2186 jiffies for Draper and 653 jiffies for Atari, versus 4025 jiffies for BASIC. The difference isn't quite as dramatic here, but for raw speed there's no denying the superiority of the Atari system over **Draper Pascal**.

```
(* Screen-Fill in Draper Pascal *)

PROGRAM SCREEN_FILL;

VAR SCREEN,I,J,TIME: INTEGER;

BEGIN

  MAXGRAPH(24); (* set up graphics *)
  GRAPHICS(24); (* mode 24         *)

  POKE(19,0); (* zero system timers *)
  POKE(20,0);

(* fetch address of screen *)

  SCREEN:=PEEK(88)+256*PEEK(89);

(* the fill loops *)

  FOR I:=0 TO 31 DO
    BEGIN
      FOR J:=0 TO 239 DO
        POKE(SCREEN+J,255);
      SCREEN:=SCREEN+240
    END;

(* fetch timer reading *)

  TIME:=PEEK(20)+256*PEEK(19);

  GRAPHICS(0);
  WRITELN(TIME,' JIFFIES')

END.
```

Listing 4.

Execution time isn't the only criteria for judging the performance of a compiler. By using DOS, I found that Draper had generated 3 sectors' worth of p-code for the **Sieve** benchmark and a scant 2 sectors for **Screen-Fill**. Compare this with the 37 sectors eaten up by Atari's **Screen-Fill** file, and a whopping 83 sectors for the **Sieve**! I can't begin to guess why **Atari Pascal** is so piggy with disk space; the documentation is silent about the inner workings of the system. We can surmise that Atari achieves at least part of their impressive speed by giving themselves plenty of elbow room.

Compilation time is yet another point of interest. Once the programs had been fully debugged, Draper's single-pass compiler took no more than ninety seconds to produce executable programs from either benchmark. **Atari Pascal**'s multi-pass compilation and linking required a good six to seven minutes for each program, including periodic disk-swaps and prompt-answerings. This comparison was made using two drives, #1 containing the system software, #2 the program source and output files.

Be ye friend or foe?

There's still one performance criteria I haven't covered in this report. It isn't often included in the high-tech comparisons they publish in *Byte* and *Dr. Dobb's Journal*, but in this case I think it makes all the difference in the world. That criteria is Development Time.

The first thing I did when I received my copy of **Draper Pascal** in the mail was to adapt and test my benchmarks. I'm not especially fluent in Pascal, and I had no prior knowledge of the system. Yet within half an hour of opening the *User's Manual*, I had a fully debugged version of **Screen-Fill** up and running. This brief experience allowed me to write **Sieve** in less than fifteen minutes.

Atari Pascal was quite another matter. It took staff programmer Charlie Bachand and myself fully *two hours* of sweat, curses and page-turning before we figured out how to implement the **Sieve** without facing a compile error. The "non-standard" GRAPHICS calls in **Screen-Fill** cost us the better part of an afternoon. These are *not* exaggerations. I should add that Charlie is fairly well-versed in Pascal and has used the Atari system before, though I can't imagine why.

How should the above comparisons be interpreted? It's clear that **Draper Pascal** isn't as fast or "complete" as Atari's **Pascal Language System**, and it's also more expensive. But when it comes down to sitting in front of a keyboard and churning out working, usable programs in a reasonable amount of time, Draper's system beats the living daylights out of the competition. If you must have Pascal for your Atari, save yourself a lot of frustration by taking a serious look at Norm Draper's long-overdue alternative. □

GRIFFIN'S LAIR

EDUCATIONAL PROGRAMS REVIEW

by Braden E. Griffin, M.D.

This month we will look at some of the available educational software in the field of mathematics. Some of this material is primarily targeted for use within a school system, and may be too expensive for many individual users. However, since Atari systems may be found in a significant number of schools (15% by some guess-timates), and some of you may have input into what system or software is purchased in your school district, I will include these programs in this review.

Unlike some other areas of education, mathematics is ideally suited for computer-assisted education. The answers are exact, usually do not require long, detailed input, and the computer can randomly generate problems and perform operations without sophisticated programming. Even a beginner can write a BASIC program which will provide a child with fundamental math drills.

The computer's ability to generate random numbers within a chosen range enables one to produce an endless stream of problems to be performed. Embellishing these programs with color, sound and unique responses ("Molly, you turkey!") allows personalization of each drill. It also helps develop elementary programming skills.

The programs reviewed here are intended for intermediate (grades 3-8) and secondary school children. For those interested in math programs for children in the early elementary age group, I refer them to **ANALOG** #14 and the review of **Mickey In The Great Outdoors** in this column, and Keith Valenza's review of **Monkeys, Math and Merriment**.

SUCCESS WITH MATH SERIES
CBS SOFTWARE
Greenwich, CT 06836
ADD/SUB/MULT./DEV. 16K C-$19.95 D-$24.95
LINEAR/QUADRATIC 48K C-$19.95 D-$24.95

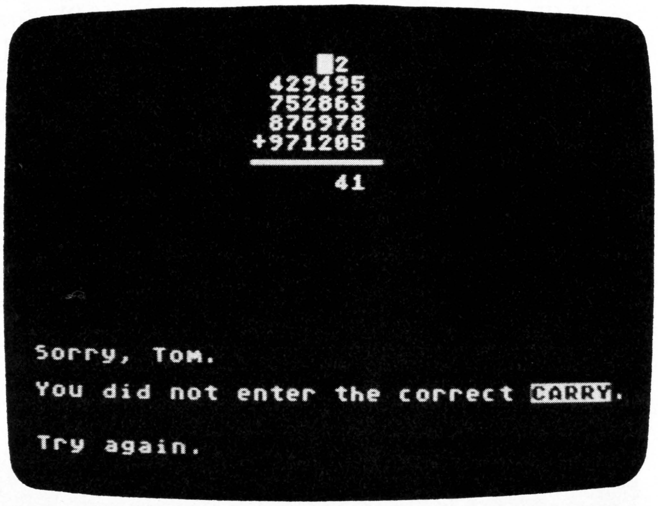
These four separate math tutorials for students from grades 1-12 are designed not only to provide math drill and practice in specific areas, but also to illustrate the step-by-step problem solving process. Unlike other math programs, the **Success With Math** series "works through" math problems, identifying errors when they occur and directing the user to the source of the error before the next step is accomplished. By using this method while prompting the student along the way, successful completion of every problem is achieved. At the end of each problem, an analysis of the errors is made.

The program takes over as soon as it is loaded and instructs the user in how to proceed. Although a small, excellently prepared manual with program

descriptions is included, the program instructions are quite clear and thorough.

Addition and Subtraction.

In the "Addition" program, any size problem up to nine numbers (rows) by nine digits (columns) may be designed. After selecting the size, a randomly generated problem is displayed. The cursor is initially found resting just below the right-hand column. The student solves the problem as if using paper and pencil by entering the sum of that column. The cursor then positions itself at the top of the next column, waiting for the student to enter the "carry." This process continues until the problem is solved. If an error is made, it is immediately called out, and a new answer may be tried. If this second answer is also wrong, the correct answer for that step is shown, and the student then continues with the solution.



Addition or Subtraction.

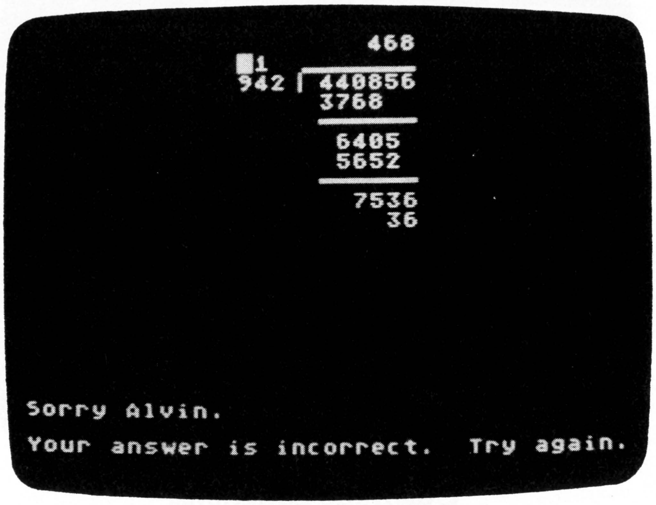
The "Subtraction" part of the program is similar to "Addition," with two exceptions. The size of the problem can only be up to five digits (only two rows, of course), and there is a choice of problems with or without borrowing. For borrowing, the letter B is entered from the keyboard. The digit to be borrowed from is then shown in inverse video and its new value is entered.

This is a very well done program which offers the student practice with large numbers and the concepts of borrowing and carrying. For grades 1-4, random computer responses and encouragements are included.

Multiplication and Division.

For grades 2-8, this program functions much like the ones above. In "Multiplication," the size of the multiplier may be up to three digits, while the multiplicand is always three digits. The multiplication process is carried out, and the final product is obtained using addition with carrying as in the previously described program.

The choice of up to three digits for the divisor in the "Division" section really gives the student extensive practice in long division. The cursor again positions itself in the specific location used, as if the problem were being done on paper. Multiplication and subtraction with borrowing are used until the final quotient is obtained.



Multiplication or Division.

Linear Equations.

Junior High students and beyond will acquire the understanding and skills necessary in basic algebraic principles with this program. All equations are in the form "AX + B = C," where A, B and C are integers. The student then solves for X using the menu of rules:

1) Add same term to both sides,
2) Subtract same term from both sides,
3) Multiply both sides by same term,
4) Divide both sides by same term,
5) Simplify both sides.

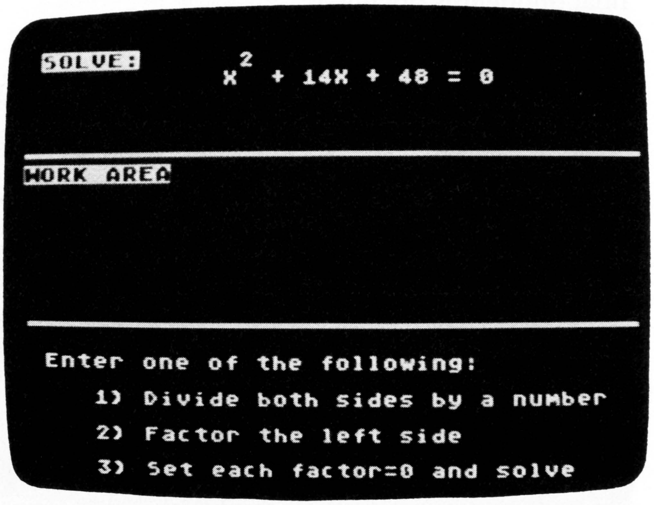
After a rule is selected, the necessary information must be provided to complete that step; e.g., if rule #3 is selected, then the user is asked what term to multiply both sides by. If the correct term is entered, the equation is then shown with the multiplier on both sides. Then rule #5 would be selected, and one would be requested to enter the new left side and, subsequently, the new right side.

As with all the programs in this series, user errors are immediately shown and explained. In this program, however, the user must supply the correct answer for each step in order for the program to continue. The student *always* succeeds in solving the equation. The program keeps track of both procedural and computational errors and displays the type and number of each at the end of each problem.

Quadratic Equations.

This program is suggested for use by students in grades 9 to 12. Factoring, setting each factor equal to

zero, and solving the separate equations are emphasized here. All equations are in the form "$AX^2 + BX + C = 0$." Two levels of difficulty may be selected, with the easier level always having the coefficient of the quadratic term (A) as one. A quadratic coefficient of two or three is used in the harder level.



Quadratic Equations.

The screen is divided into three areas: the original equation, a work area, and a message/instruction area. A menu appears with the following options:

1) Divide both sides by a number,
2) Factor the left side,
3) Set each factor equal to 0 and solve.

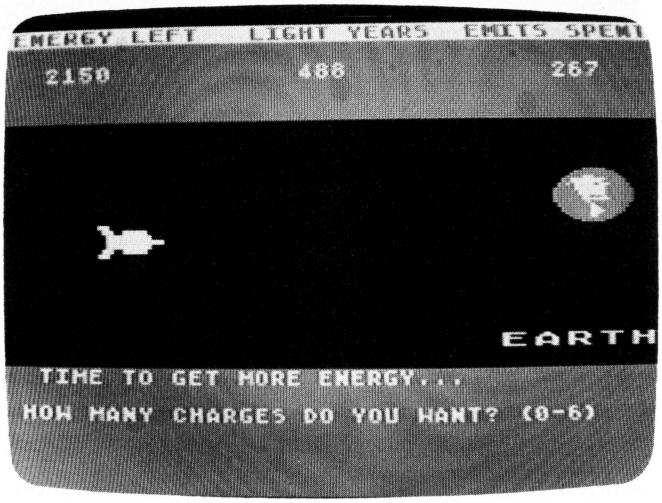
After the equation is simplified, the user enters the appropriate factors in the work area. If they are incorrect, an error message appears along with the product of the incorrect factors. A new pair of factors may then be entered. If these are wrong, the program goes to a tutorial section, explaining in detail how to factor the quadratic expression. When the correct factors have been entered, they are set equal to zero. A new menu appears and each equation is solved for X as in the "Linear Equations" program. The same process of error-tracking is also performed.

All four of these programs are excellent. The explanations are comprehensive and clear, and the positive reinforcement is skillfully accomplished. My eighth grader was working with the "Linear Equations" program one evening before I had looked at it myself. I was shocked when I saw the computer respond with "You only made one mistake, FOOL." I soon found out that, at the start of each program, the user's name is requested. . .and my young court jester had rechristened himself "FOOL." All fooling aside, these four programs not only provide practice in math concepts, they actually *teach* them. The producers of these packages are right on target when they state "There's no greater motivation for learning than success."

GALAXY II MATH FACTS
RANDOM HOUSE SCHOOL DIVISION
400 Hahn Road
Westminster, MD 21157
48K Disk, $49.50 each ($174.00/set)

As the commander of a spaceship, you must equip and fly your ship through uncharted regions of space, avoiding Black Holes, fighting Klingons, maneuvering dense star fields, and finally returning to your home base, Earth. In order to accomplish a specific task, answers to a variety of math facts must be supplied. There are six programs available in the Galaxy II series, offering drill and practice in an assortment of math concepts.

The same game format is used in all programs. At the beginning of the program, the level of difficulty (1-6) must be selected. The response time is then chosen from four options, ranging from Beginner rank (7 seconds) to Captain (4 seconds). Players begin 2000 light years from Earth. The time elapsed, energy spent and distance from Earth are displayed at the top of the screen throughout the game. The engines must be loaded and fired to begin the return mission. Each of these functions requires successfully answering math problems. If the answer is wrong or too late, a second chance is given. If the second response is also incorrect, the correct answer is supplied, and that particular function will not be performed. This method is also used to arm the ship with torpedos and space bombs.

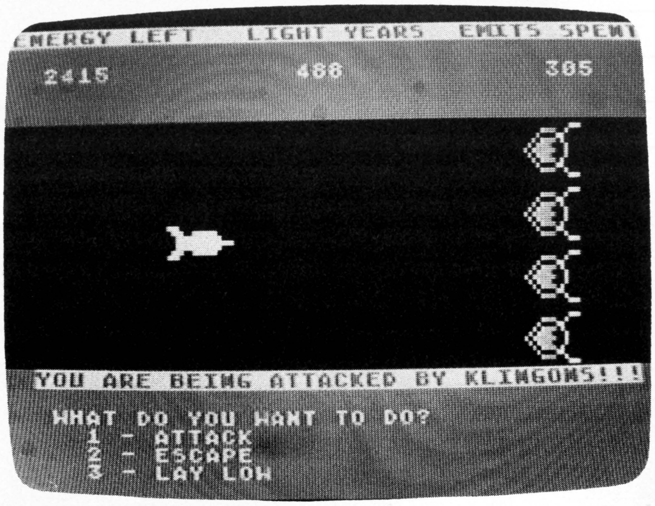


Galaxy II Math Facts.

"I need more power, Scotty!"

During the game, a player must replenish the ship's energy supplies. Three potential disasters are encountered during the mission. Approaching a Black Hole, a math problem appears on the screen. A correct answer causes the spaceship to move farther from the gravitational force of the Black Hole. An incorrect response, or unsolved problem, moves the

ship closer to it. If the ship is pulled into the Black Hole, it is destroyed, and the game ends. Similarly, correct responses enable one to fight Klingons and avoid destruction in a star field. After avoiding all hazards, the final destination is reached and the flight time is displayed. The goal is to record the fastest flight time possible. It is not a simple task to reach Earth. Even if every problem is correctly solved, some gamesmanship skills are required.



Galaxy II Math Facts.

Basic Facts.

The first of the six programs in the series provides a drill in automatically recalling the basic facts commonly mastered in the first through the fourth grades. "Basic Facts" offers six levels of progress, from addition and subtraction to multiplication and division, and uses number facts from 0-5 up to 0-12.

Place Value.

Here the student must identify the position or place value of a digit within a given number. The numbers range from hundred thousands to thousandths. The recommended grade level for this program is from 1-6.

Fractions I and II.

These programs are presented on two disks. They include the concepts of comparing fractions, simplifying fractions, changing improper fractions to mixed numbers and vice versa, and performing basic math operations that use fractions. In "Fractions I," a true (T) or false (F) response is required. In "Fractions II," the numerator and denominator are sequentially entered. Fourth through sixth grade level is necessary to use these programs.

Rounding and Estimating.

The first three levels of this program require the student to round numbers ranging from thousandths to thousands. The last three levels stress estimation of the sum or difference of two numbers. Recommended grade level is 3-6.

Decimals.

Frequently a difficult concept for some students, this section provides drills in comparing decimals, changing from percentages to decimals and vice versa, and solving problems containing decimals. Fifth through seventh graders will benefit from this very well done program.

Integers.

Emphasis on the properties of integers for students in grades six through eight is found here. The four basic math operations are used as a foundation for students before they advance to algebra, e.g., $-6<-4$ (T or F) or $-42/7 = ?$.

Galaxy II Math Facts is not designed to teach concepts, but to provide drill and practice in the different concepts. It does this quite well. The game format and graphic displays make for an enjoyable educational experience. An excellent manual is included. The programs may be purchased individually or in a package at reduced cost.

SURVIVAL MATH
SUNBURST COMMUNICATIONS, Inc.
Pleasantville, NY 10570
16K Disk $50.00

Survival Math sounds like a cutesy arcade-style game where one shoots down invading aliens by answering math problems. Wrong again, Pythagorasbreath. This is the most practical, realistic and func tional educational software package for the development of mathematical skills I have seen. There are no fancy graphics. The straight-forward text format is all that is necessary for this set of programs, designed for students in grades 7-12. Any student who successfully manages these four simulations will be well prepared for the outside world.

Travel Agent Contest.

Given a limit for expenses, a seven-day and six-night trip to Lake Geneva is to be arranged. One must try to come as close to this limit as possible without exceeding it. Expenses for travel, food, lodging and entertainment must be taken into account. As each expense category is selected, several options are given. One may decide to travel by plane, bus or train. It is more expensive to travel by plane, but money can be saved on meals during travel if it is selected. One-way fares are provided and, after making a choice, the cost of the round trip must be calculated. If the figure entered is incorrect, it is noted to be either too high or too low, and the total cost is again requested. When the exact cost is entered, the next expense category is planned.

In each category, a number of options is available. Just like in the real world, one may go "first class" and stay in a posh hotel, but then may have to eat at

(Continued on page 29)

Turn your Atari into a Ferrari.

Introducing the all-new 1984 Indus GT™ disk drive. The most advanced, most complete, most handsome disk drive in the world.

A flick of its "Power" switch can turn your Atari into a Ferrari.

Looks like a Ferrari.
The Indus GT is only 2.65" high. But under its front-loading front end is slimline engineering with a distinctive European-Gran flair.

Touch its LED-lit CommandPost™ function control AccuTouch™ buttons. Marvel at how responsive it makes every Atari home computer.

Drives like a Rolls.
Nestled into its soundproofed chassis is the quietest and most powerful disk drive power system money can buy. At top speed, it's virtually unhearable. Whisper quiet.

Flat out, the GT will drive your Atari track-to-track 0-39 in less than one second. And when you shift into SynchroMesh DataTransfer,™ you'll increase your Atari's baud rate an incredible 400%. (Faster than any other Atari system drive.)

And, included as standard equipment, each comes with the exclusive GT DrivingSystem™ of software programs. World-class word processing is a breeze with the GT Estate WordProcessor.™ And your dealer will describe the two additional programs that allow GT owners to accelerate their computer driving skills.

Also, the 1984 Indus GT is covered with the GT PortaCase.™ A stylish case that conveniently doubles as a 80-disk storage file.

Parks like a Beetle.
The GT's small, sleek, condensed size makes it easy to park.

And its low $449 price makes it easy to buy.

So see and test drive the incredible new 1984 Indus GT at your nearest computer dealer soon.

The drive will be well worth it.

INDUS™

The all-new 1984 Indus GT Disk Drive.
The most advanced, most handsome disk drive in the world.

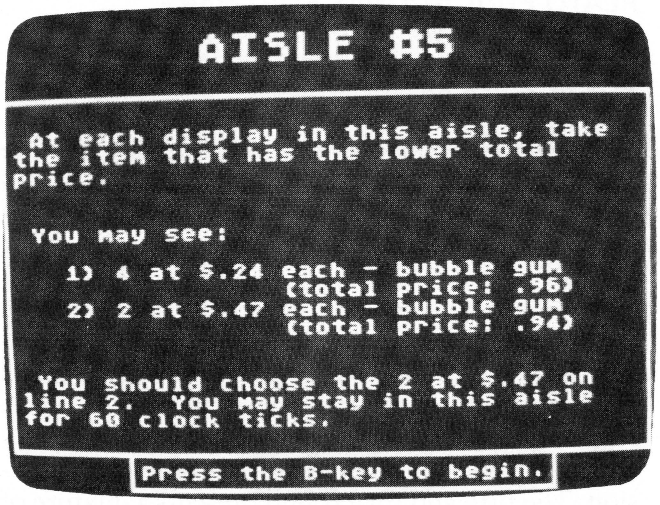
For additional information, call 1-800-33-INDUS. In California, call 1-800-54-INDUS. 213/882-9600.

the supermarket. If there is enough money, special events may be attended with or without a friend. Now, who wants to go to a ball game alone?

It should take about five to ten minutes to prepare a plan. The basic math skills required include working with money/decimals and estimation. Besides the computational skills, this program provides students with practice in planning a course of action within a budget. Each time the program is run, the maximum expense allowed is different, as are the costs of the various options within each expense category.

Smart Shopper Marathon.

Remember the supermarket contests where a woman with a shopping cart was given ten minutes to fill it with free groceries? This program recreates that scenario with a few exceptions. There are five aisles in this supermarket, each with twenty-five displays. At the beginning of each aisle, a set of instructions is given, The correct choice must be made at each display based on these instructions. Unlike the other three programs in this series, this one has a time limit. One is given sixty clock ticks for each aisle. If an incorrect response is made, ten ticks are deducted. One point is scored for each correct answer, and a 5-point bonus is awarded for completing all 25 questions.



Smart Shopper Marathon.

A different math skill is required in each aisle. For example, one may be required to pick the item with the lower unit price. Two items are described: 1) Pay $0.60 for two bars of soap; 2) Pay $1.10 for three bars of soap. The #1 key would be pressed here, and the next display would be described. In this example, skill in estimation of division of a decimal by a whole number is required. Estimation of the product of two whole numbers is used when determining the greater total weight of two items in another aisle. Calculating the bigger dollar savings of two items, selecting the item with the larger percent savings, and deciding which items have the lower total price are the objectives in the other three aisles. The order of the aisles changes with each new run. The item prices also vary, but they are always in a realistic price range.

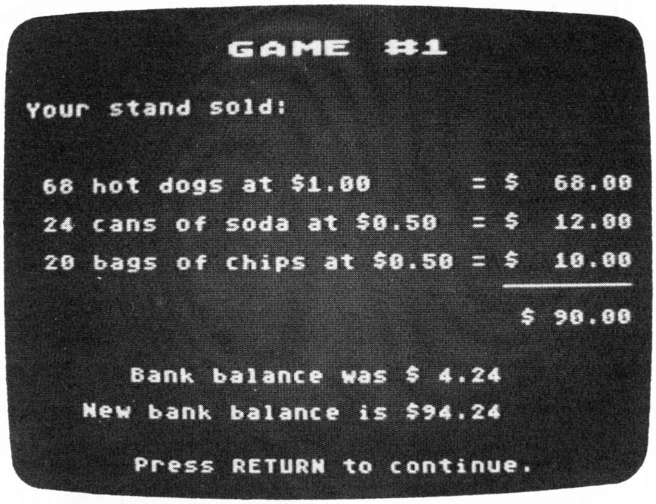
Careful estimates — not exact computations — are stressed in "Marathon." Both speed and accuracy are important since, if every item price is calculated, the student will not have as many opportunities to score points. Comparison shopping is frequently encouraged; the number of individuals who lack experience in this skill is amazing.

Hot Dog Stand.

This program is so realistic that a friend of ours wanted everybody responsible for the local Little League concession stand to practice it before next season. The student begins with $200 and must purchase hot dogs, buns, chips, soda and courtesy kits (napkins, ketchup, etc.) for their stand for an 8-game football season. Before each game, the prices of the items are set. The object is to earn $2,500 by the end of the season.

(continued on next page)

In order to make this simulation more realistic, a number of variables exist. The stands can only hold one thousand fans. Obviously, the crowd size has a direct bearing on sales. The weather affects crowd size and consumption (more soda will be sold on a sunny day). A weather forecast is given; however, the actual weather at game time may vary. Other factors which may alter the crowd size include the type of game (homecomings attract the biggest crowds), time of day, and day of the week (Saturday crowds are larger). Hot dogs cannot be sold without buns, so the purchase of these two items should always match. If there is an inadequate supply of courtesy kits, the sale of hot dogs decreases. Chips, soda and kits do not spoil, but any hot dogs and buns not sold each week are lost. There is also a choice in the quality; inexpensive, low quality hot dogs do not sell as well.



Hot Dog Stand.

Pricing is very important. Not too many people will buy $2.50 hot dogs, and a shirtless concessionaire is the result of $0.10 bargain-priced dogs. Unit pricing varies, and it is probably wise to stock up on certain non-perishable items early in the year.

Each time the program runs, the football schedule and other variables differ. After each game, the sales figures for each item, the total sales, and the new bank balance are shown. This simulation provides students with practice in unit pricing, price setting, inventory assessment and dealing with the concepts of profit and loss. Sounds like important stuff to me.

Foreman's Assistant.

In order to build a playroom, six different jobs must be completed. The object of this program is to successfully estimate the amount of materials required for each. There is a $50 limit on unused materials. Also, the entire job must be completed within thirty days. If an unacceptable materials estimate is made (too little material, for example), more days are required to finish the project. After each task is described, the total amount and cost of the materials must be calculated.

The tasks faced in this project are very realistic. The dimensions of the playroom are given. One must calculate the necessary number of vertical supports, 4-by-8 wall panels, bricks to cover one wall (dimensions given for each brick), and gallons of paint to give the room two coats. The number of feet of shelf stock required to build bookshelves on one wall must be computed. Finally, the floor is to be covered with 9-inch tiles.

This is the most difficult of the four programs, and a calculator might come in handy. Along with the obvious math skills required, "Foreman's Assistant" provides practice with conversion of measurments. It is suggested that students complete their estimates using their own methods and that established formulae not be provided. Most students apparently use a variety of techniques without generation of a formula.

In summary, the four simulations in **Survival Math** are truly educational. The fundamental math skills we expect our children to learn can all be found in this package. Challenging and realistic, I highly recommend **Survival Math**.

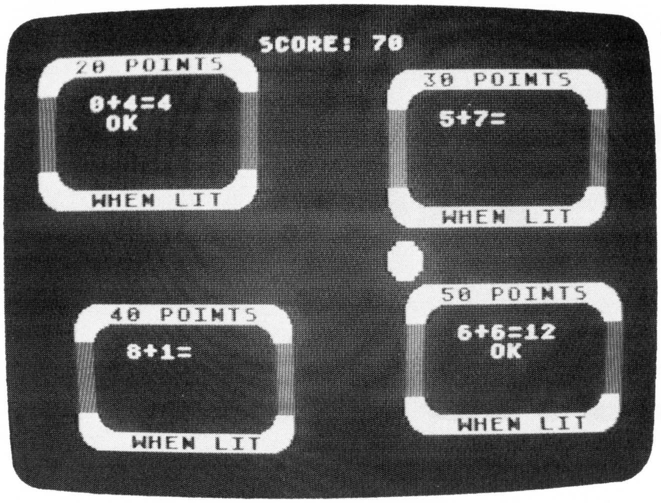
WHOLE NUMBERS
PLATO EDUCATIONAL SOFTWARE
P.O. Box 261127
San Diego, CA 92126
48K Disk $60.00

Using the format of the pinball game, **Whole Numbers** provides intermediate-grade level students with drills in the four basic arithmetic operations which are both fun and challenging. Four bumpers are displayed on the screen, each containing a randomly generated problem. As one begins the game, an animated ball moves about the screen. When the ball touches a bumper, the answer to its enclosed problem must be typed in. There are five rounds per game, the first four dealing separately with addition, subtraction, multiplication, and division. The fifth and final round presents a mixture of problems from the first four groups.

The student's goal is to accumulate points, which are awarded for correct answers. Each bumper has a different point value and as each round progresses, the time allotted for a response becomes shorter and shorter. After each correct response, the ball randomly moves to another bumper, and a new problem is written inside. If the correct answer is not supplied within three attempts, or before the time runs out, that particular bumper ceases to function for that round. If three of the four bumpers are turned off, the ball exits the screen and the next round begins.

Quick responses are necessary to continue amassing points in each round. Eventually, the time limit

becomes so short that it is impossible to enter the answer. A running point total is shown at the top of the screen, and after the last round, if the cumulative point total is one of the ten highest, the student's score and his or her initials are entered into a Hall of Fame.



Whole Numbers.

Whole Numbers drills students in the basic number facts from 0 to 10. It is designed to improve a student's basic arithmetic skills by stressing speed and accuracy. The final round gives practice in distinguishing between the operations. Competition with other students, or just trying to improve one's own score, makes each game exciting. The accompanying manual includes a section on teaching strategies and different activities for individual or group use. Entries into the Hall of Fame may be deleted selectively by using a password found in the manual. This option is particularly useful if one individual is dominating the high scores, or at the beginning of a new class.

A fast-paced and challenging approach to arithmetic drills, **Whole Numbers** is equally appropriate for use in the home or classroom. □

A BINARY FILE MENU/LOADER

32K Disk

by Richard J. Kalagher

More and more programs for the Atari are being written in machine language. **ANALOG** in particular has been a leader, publishing many games and utilities in the Atari's mother tongue. Machine language programs run much faster and, in general, the games are much smoother and playable. The only nuisance is loading a machine language program. First you must boot DOS, then type "L" and press RETURN, then type the name of the file (spelling it right) and press RETURN. Children, especially, find this complicated to remember.

When the Atari was first released, most of the programs were written in BASIC. To make loading these BASIC programs easier, many self-booting menu programs were written. The user just had to boot the disk and press a letter or number from a menu in order to load and run the program.

The program described in this article does the same thing for machine language programs. To use it, create an AUTORUN.SYS file by typing in and running the BASIC program. The file will automatically be created. You can also type in the assembly language program and use an assembler to create the file.

Now, just transfer the AUTORUN.SYS file to your disks which contain binary load files. You can erase the DUP.SYS file but you need the DOS.SYS file. When the disk boots, the menu will list up to 26 files on the disk, omitting any with a "SYS" extension. Any other extension will be ignored. Thus you should not put two files with the same name and a different extension on the same disk. Also, no checks are made to see if the file is a valid binary load file. You should make sure the file will load from DOS with the "L" option before transferring it to your menu disk.

To load and run a binary load program, just press the letter corresponding to the program you want. The program should load and execute. In a very few cases, the program may not load properly. This is usually due to the program overwriting the loading routine. If you have the assembler version of the program, try assembling the program at a different origin.

The program itself is straightforward. I try to write assembly programs in a series of short subroutines. This makes it much easier to follow the code and helps in changing and debugging. The subroutine calls between lines 420 and 680 constitute the main program. If you understand how to call the Atari operating system and how binary files are formatted, you should have no trouble following the code and adapting it to your own needs. Also notice that the code is somewhat backwards. The subroutines to load a file are first, followed by the subroutines to display the menu. I did this because some programs will overwrite the latter portion of the code, which will not cause a problem since this code is not needed for the loading process.

Finally, I could have used the loader in DOS or at least copied the code. There are two reasons I didn't, however. First, the loading routine is independent of any changes in DOS. Second, I did not have a copy of the DOS code at the time I wrote the program. I also learned a lot more by doing it myself. □

(Listing starts on p. 34.)

BASIC Listing.

```
100 REM BINARY FILE MENU/LOADER
110 REM BASIC PROGRAM TO CREATE
120 REM AN AUTORUN.SYS FILE
130 REM LOADING AT $1DFC
140 REM
150 REM BY RICHARD J. KALAGHER
160 OPEN #1,8,0,"D:AUTORUN.SYS"
180 DIM A(10):TRAP 270
190 FOR N=1000 TO 1800 STEP 10
200 PRINT "READING LINE ";N:FOR J=1 TO
 10
210 READ X:A(J)=X:TOT=TOT+X
220 NEXT J
230 READ X:IF X<>TOT THEN 270
240 FOR J=1 TO 10:PUT #1,A(J):NEXT J
250 NEXT N
260 END
270 PRINT "ERROR IN LINE ";N:END
1000 DATA 255,255,124,29,255,29,32,224
,30,32,1265
1010 DATA 37,31,32,224,29,32,193,30,48
,6,1927
1020 DATA 32,85,31,76,133,29,32,187,31
,32,2595
1030 DATA 182,30,32,177,30,32,148,32,3
2,246,3536
1040 DATA 31,32,115,32,32,177,30,32,61
,32,4110
1050 DATA 32,21,30,32,78,30,32,213,29,
32,4639
1060 DATA 147,30,32,125,30,32,78,30,32
,136,5311
1070 DATA 30,32,98,30,32,165,30,48,9,3
2,5817
1080 DATA 203,29,76,171,29,108,226,2,3
2,182,6875
1090 DATA 30,108,224,2,96,169,212,141,
226,2,8085
1100 DATA 169,29,141,227,2,96,162,32,1
69,3,9115
1110 DATA 157,66,3,169,6,157,74,3,169,
250,10169
1120 DATA 157,68,3,169,29,157,69,3,32,
86,10942
1130 DATA 228,96,68,49,58,42,46,42,18,
30,11619
1140 DATA 158,32,155,0,0,162,32,169,3,
157,12487
1150 DATA 66,3,169,0,157,68,3,169,30,1
57,13309
1160 DATA 69,3,169,4,157,74,3,32,86,22
8,14134
1170 DATA 96,162,32,169,7,157,66,3,165
,192,15183
1180 DATA 157,68,3,165,193,157,69,3,16
5,198,16361
1190 DATA 157,72,3,165,199,157,73,3,32
,86,17308
1200 DATA 228,96,169,19,133,192,169,30
,133,193,18670
1210 DATA 169,2,133,198,169,0,133,199,
32,47,19752
1220 DATA 30,96,165,196,56,229,194,133
,198,165,21214
1230 DATA 197,229,195,133,199,24,165,1
98,105,1,22660
1240 DATA 133,198,165,199,105,0,133,19
9,96,173,24061
1250 DATA 19,30,133,194,173,20,30,133,
195,96,25084
1260 DATA 173,19,30,133,196,173,20,30,
133,197,26188
1270 DATA 96,174,19,30,232,240,1,96,17
4,20,27270
1280 DATA 30,232,240,1,96,32,78,30,96,
165,28270
1290 DATA 194,133,192,165,195,133,193,
32,47,30,29584
1300 DATA 96,162,48,76,184,30,162,32,1
69,12,30555
1310 DATA 157,66,3,32,86,228,96,162,32
,169,31586
1320 DATA 5,157,66,3,169,0,157,68,3,16
9,32383
1330 DATA 30,157,69,3,169,18,157,72,3,
169,33230
1340 DATA 0,157,73,3,32,86,228,96,162,
48,34115
1350 DATA 169,3,157,66,3,169,4,157,68,
3,34914
1360 DATA 169,31,157,69,3,169,8,157,74
,3,35754
1370 DATA 169,0,157,75,3,32,86,228,169
,1,36674
1380 DATA 141,240,2,96,69,58,162,48,16
9,11,37670
1390 DATA 157,66,3,165,194,157,68,3,16
5,195,38843
1400 DATA 157,69,3,165,198,157,72,3,16
5,199,40031
1410 DATA 157,73,3,32,86,228,96,169,1,
133,41009
1420 DATA 84,169,10,133,85,169,67,133,
194,169,42222
1430 DATA 31,133,195,169,16,133,198,16
9,0,133,43399
1440 DATA 199,32,6,31,230,84,96,66,105
,110,44358
1450 DATA 97,114,121,32,76,111,97,100,
32,77,45215
1460 DATA 101,110,117,65,0,32,153,31,3
2,177,46033
1470 DATA 31,169,0,133,194,169,30,133,
195,169,47256
1480 DATA 10,133,198,169,0,133,199,230
,84,160,48572
1490 DATA 6,173,83,31,201,78,144,12,16
0,24,49484
1500 DATA 165,84,201,16,208,4,169,3,13
3,84,50551
1510 DATA 132,85,173,83,31,141,0,30,16
9,45,51440
1520 DATA 141,1,30,32,18,32,32,6,31,23
8,52001
1530 DATA 83,31,96,173,10,30,201,83,20
8,16,52932
1540 DATA 173,11,30,201,89,208,9,173,1
2,30,53868
1550 DATA 201,83,208,2,104,104,96,173,
1,30,54870
1560 DATA 201,32,240,2,104,104,96,169,
20,133,55971
1570 DATA 84,169,4,133,85,169,215,133,
194,169,57326
1580 DATA 31,133,195,169,31,133,198,16
9,0,133,58518
1590 DATA 199,32,6,31,96,80,114,101,11
5,115,59407
1600 DATA 32,116,104,101,32,76,101,116
,116,101,60302
1610 DATA 114,32,111,102,32,89,111,117
,114,32,61156
1620 DATA 67,104,111,105,99,101,162,48
,169,3,62125
1630 DATA 157,66,3,169,16,157,68,3,169
,32,62965
1640 DATA 157,69,3,169,4,157,74,3,32,8
6,63719
1650 DATA 228,96,75,58,160,0,174,84,31
,177,64802
1660 DATA 194,9,128,157,159,32,232,200
,200,177,66290
1670 DATA 194,157,159,32,201,32,240,11
,232,200,67748
1680 DATA 192,10,144,241,169,32,157,15
9,32,142,69026
1690 DATA 84,31,238,84,31,96,0,162,0,1
89,69941
1700 DATA 159,32,232,205,60,32,208,247
,169,68,71353
1710 DATA 141,0,30,169,58,141,1,30,160
,0,72083
1720 DATA 189,159,32,153,2,30,232,200,
201,32,73313
1730 DATA 208,244,136,169,46,153,2,30,
200,169,74670
1740 DATA 42,153,2,30,200,169,32,153,2
,30,75483
1750 DATA 96,162,48,169,7,157,66,3,169
,0,76360
```

```
1760 DATA 157,72,3,157,73,3,32,86,228,
201,77372
1770 DATA 65,144,234,205,83,31,176,229
,9,128,78676
1780 DATA 141,60,32,96,169,212,141,224
,2,169,79922
1790 DATA 29,141,225,2,96,226,2,227,2,
124,80996
1800 DATA 29,0,0,0,0,0,0,0,0,0,81025
```

•

CHECKSUM DATA
(see p. 15)

```
100 DATA 529,420,820,163,86,66,286,854
,647,199,879,738,789,497,755,7728
260 DATA 46,323,373,838,117,111,56,805
,919,626,141,134,965,917,121,6492
1130 DATA 70,935,133,901,220,430,6,748
,416,742,755,731,273,431,165,6956
1280 DATA 911,708,405,180,911,963,135,
830,193,126,170,452,425,957,496,7862
1430 DATA 495,118,403,357,479,495,192,
167,156,801,140,167,137,214,494,4815
1580 DATA 491,401,634,407,269,16,861,1
88,689,700,699,109,483,29,249,6225
1730 DATA 463,956,186,949,334,264,969,
862,4983
```

•

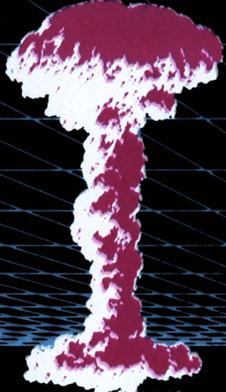
Assembly language listing

```
; BINARY LOAD MENU PROGRAM
; NAME OBJECT FILE D:AUTORUN.SYS
;
; RICHARD J. KALAGHER
;
; EQUATES
;
SL      =       $CØ     ;PAGE ZERO TEMPS
SH      =       $C1
STL     =       $C2
STH     =       $C3
ENL     =       $C4
ENH     =       $C5
BLL     =       $C6
BLH     =       $C7
CIO     =       $E456
ICCMD   =       $Ø342   ;COMMAND
ICBAL   =       $Ø344   ;BUFF ADDR LOW
ICBAH   =       $Ø345   ;BUFF ADDR HIGH
ICAX1   =       $Ø34A   ;AUX 1
ICAX2   =       $Ø34B   ;AUX 2
ICBLL   =       $Ø348   ;BUFF LEN LOW
ICBLH   =       $Ø349   ;BUFF LEN HIGH
ROWCRS  =       84      ;SCREEN ROW
COLCRS  =       85      ;SCREEN COLUMN
CRSINH  =       $Ø2FØ   ;CURSOR INHIBIT
;
        *=      $1D7C   ;***SEE TEXT***
;
MENU    JSR OPENSC      ;OPEN SCREEN
        JSR HEADER      ;TITLE ON SCREEN
        JSR OPEDIR      ;OPEN DIRECTORY
GG      JSR GETDIR      ;GET DIRECTORY ENTRY
        BMI FOOT        ;DIRECTORY DONE? YES.
        JSR DIRSCN      ;PUT ENTRY ON SCREEN
        JMP GG          ;GET ANOTHER
;
FOOT    JSR FOOTER      ;CHOOSE MESSAGE
        JSR CLOSE2      ;CLOSE DIRECTORY
        JSR CLOSE3      ;CLOSE SCREEN
        JSR SETUP       ;SET RUN+INIT TO RTS
        JSR OPENK       ;OPEN KEYBOARD
        JSR GETLET      ;GET A LETTER
        JSR CLOSE3      ;CLOSE KEYBOARD
        JSR FINDNA      ;FIND FILE NAME
        JSR OPEFIL      ;OPEN THE FILE
GETFIL  JSR READ2       ;GET TWO BYTES
        JSR INIT        ;SET INIT DEFAULT
        JSR CHKFF       ;CHECK IF HEADER
        JSR STRAD       ;PUT START ADDR
        JSR READ2       ;TWO MORE BYTES
        JSR ENDAD       ;PUT END ADDR
        JSR BUFLEN      ;COMPUTE BUFFER LEN
        JSR GETDAT      ;GET DATA RECORD
        BMI JSTART      ;START PROGRAM IF EOF
        JSR JINIT       ;TRY TO INITIALIZE
        JMP GETFIL      ;DO NEXT SEGMENT
;
JINIT   JMP ($Ø2E2)     ;INITIALIZE
JSTART  JSR CLOSE2      ;CLOSE THE FILE
        JMP ($Ø2EØ)     ;START PROGRAM
;
R       RTS             ;FOR INIT CODE
;
; START OF SUBROUTINES
;
; SET INITIALIZATION TO DEFAULT
;
INIT    LDA #R&$FF
        STA $Ø2E2
        LDA #R/256
        STA $Ø2E3
        RTS
;
; OPEN THE DIRECTORY FILE
;
OPEDIR  LDX #$2Ø        ;IOCB 2
        LDA #3          ;OPEN COMMAND
        STA ICCMD,X
        LDA #6          ;DIRECTORY
        STA ICAX1,X
        LDA #B1&$FF
        STA ICBAL,X
        LDA #B1/256
        STA ICBAH,X
        JSR CIO
        RTS
;
B1      .BYTE "D1:*.*"
FNAME   *=  *+18        ;FILENAME BUFFER
        .BYTE $9B
BAL     .BYTE Ø         ;TWO BYTE BUFFER FOR
BAH     .BYTE Ø         ;FILE HEADER AND ADDR
;
; OPEN FILE FOR READING
;
OPEFIL  LDX #$2Ø
        LDA #3
        STA ICCMD,X
        LDA #FNAME&$FF
        STA ICBAL,X
        LDA #FNAME/256
        STA ICBAH,X
        LDA #4          ;OPEN FOR READ
        STA ICAX1,X
        JSR CIO
        RTS
;
; GET A SEGMENT FROM THE FILE
;
GETREC  LDX #$2Ø
        LDA #7          ;GET BYTES
        STA ICCMD,X
        LDA SL
        STA ICBAL,X
        LDA SH
        STA ICBAH,X
        LDA BLL
        STA ICBLL,X
        LDA BLH
        STA ICBLH,X
        JSR CIO
        RTS
;
; READ TWO BYTES INTO BUFFER $CØ
;
READ2   LDA #BAL&$FF
        STA SL
        LDA #BAL/256
        STA SH
        LDA #2
        STA BLL
        LDA #Ø
        STA BLH
        JSR GETREC      ;GO GET THEM
        RTS
;
; CALCULATE BUFFER LENGTH
;
BUFLEN  LDA ENL
        SEC
        SBC STL
        STA BLL         ;LOW BYTE
        LDA ENH
        SBC STH
        STA BLH         ;HIGH BYTE
        CLC             ;NOW ADD ONE
        LDA BLL
        ADC #1
        STA BLL
        LDA BLH
        ADC #Ø          ;TAKE CARE CARRY
        STA BLH
        RTS
;
; PUT START ADDRESS
;
STRAD   LDA BAL
        STA STL
        LDA BAH
        STA STH
        RTS
;
; DO SAME FOR END ADDRESS
;
ENDAD   LDA BAL
        STA ENL
        LDA BAH
        STA ENH
        RTS
;
; CHECK IF HEADER
;
CHKFF   LDX BAL
        INX             ;MAKE Ø IF $FF
        BEQ TEST2       ;TEST NEXT BYTE
        RTS             ;NOT A HEADER
TEST2   LDX BAH
        INX
        BEQ ITSFF       ;IT'S $FF
        RTS
ITSFF   JSR READ2       ;TWO MORE BYTES
        RTS
;
; GET DATA BYTES
;
GETDAT  LDA STL
        STA SL
        LDA STH
        STA SH
        JSR GETREC
        RTS
;
; CLOSE IOCBS
;
CLOSE3  LDX #$3Ø
        JMP CLB
;
CLOSE2  LDX #$2Ø
CLB     LDA #$ØC
        STA ICCMD,X
        JSR CIO
        RTS
;
; GET A DIRECTORY RECORD
;
GETDIR  LDX #$2Ø        ;IOCB 2
        LDA #5          ;GET BYTES
        STA ICCMD,X
        LDA #FNAME&$FF
        STA ICBAL,X
        LDA #FNAME/256
        STA ICBAH,X
        LDA #18         ;DIR ENTRY LENGTH
        STA ICBLL,X
```

```
        LDA #Ø
        STA ICBLH,X
        JSR CIO
        RTS
;
; OPEN THE SCREEN
;
OPENSC  LDX #$3Ø
        LDA #3      ;OPEN
        STA ICCMD,X
        LDA #B4&$FF
        STA ICBAL,X
        LDA #B4/256
        STA ICBAH,X
        LDA #8      ;WRITE
        STA ICAX1,X
        LDA #Ø      ;MODE Ø
        STA ICAX2,X
        JSR CIO
        LDA #1      ;ANY NONZERO
        STA CRSINH  ;INHIBIT CURSOR
        RTS
;
B4      .BYTE "E:"  ;SCREEN DEVICE
;
; WRITE TO THE SCREEN
;
WRITES  LDX #$3Ø
        LDA #$ØB    ;PUT CHARACTERS
        STA ICCMD,X
        LDA STL
        STA ICBAL,X
        LDA STH
        STA ICBAH,X
        LDA BLL
        STA ICBLL,X
        LDA BLH
        STA ICBLH,X
        JSR CIO
        RTS
;
; PUT HEADER ON SCREEN
;
HEADER  LDA #1
        STA ROWCRS  ;ROW 1
        LDA #1Ø
        STA COLCRS  ;COLUMN 1Ø
        LDA #B5&$FF
        STA STL
        LDA #B5/256
        STA STH
        LDA #16     ;BUFFER LENGTH
        STA BLL
        LDA #Ø
        STA BLH
        JSR WRITES
        INC ROWCRS  ;LEAVE A SPACE
        RTS
;
B5      .BYTE "Binary Load Menu"
NUMKEY  .BYTE "A"
OFFSET  .BYTE Ø     ;FILE NAME OFFSET
;
; WRITE DIRECTORY ENTRIES ON SCR
;
DIRSCN  JSR CHKSYS  ;CHECK "SYS" EXT
        JSR CHKFRE  ;CHECK "FREE SECTOR"
        LDA #FNAME&$FF
        STA STL
        LDA #FNAME/256
        STA STH
        LDA #1Ø     ;USE NAME, NOT EXT
        STA BLL
        LDA #Ø
        STA BLH
        INC ROWCRS
        LDY #6
        LDA NUMKEY
        CMP #78
        BCC FIRCOL
        LDY #24     ;SECOND COLUMN
        LDA ROWCRS
        CMP #16
        BNE FIRCOL
        LDA #3
        STA ROWCRS
FIRCOL  STY COLCRS
        LDA NUMKEY  ;LETTER OF ENTRY
        STA FNAME   ;PUT IN BUFFER
        LDA #45     ;DASH
        STA FNAME+1
        JSR PUTBUF  ;NAME INTO BUFF
        JSR WRITES  ;PUT ON SCREEN
        INC NUMKEY  ;ADVANCE FOR NEXT
        RTS
;
; CHECK IF SYS FILE
;
CHKSYS  LDA FNAME+1Ø
        CMP #83     ;S
        BNE NOTSYS
        LDA FNAME+11
        CMP #89     ;Y
        BNE NOTSYS
        LDA FNAME+12
        CMP #83     ;S
        BNE NOTSYS  ;SYS FILE? NO.
        PLA         ;YES-PULL RETURN ADDR
        PLA         ;DON'T WRITE TO "E:"
NOTSYS  RTS
;
; CHECK IF "FREE SECTOR" ENTRY
;
CHKFRE  LDA FNAME+1
        CMP #$2Ø    ;IT SHOULD BE SPACE
        BEQ NOTFRE
        PLA
        PLA         ;SKIP REST OF SCREEN
NOTFRE  RTS
;
; WRITE PROMPT MESSAGE ON SCREEN
;
FOOTER  LDA #2Ø
        STA ROWCRS
        LDA #4
        STA COLCRS
        LDA #B6&$FF
        STA STL
        LDA #B6/256
        STA STH
        LDA #31
        STA BLL
        LDA #Ø
        STA BLH
        JSR WRITES
        RTS
;
B6      .BYTE "Press the Letter"
        .BYTE " of Your Choice"
;
; OPEN KEYBOARD
;
OPENK   LDX #$3Ø
        LDA #3      ;OPEN
        STA ICCMD,X
        LDA #B7&$FF
        STA ICBAL,X
        LDA #B7/256
        STA ICBAH,X
        LDA #4      ;READ
        STA ICAX1,X
        JSR CIO
        RTS
;
B7      .BYTE "K:"  ;KEYBOARD DEVICE
;
; PUT FILE NAMES IN A BUFFER
;
PUTBUF  LDY #Ø
        LDX OFFSET  ;CURRENT LOCATION
        LDA (STL),Y
        ORA #$8Ø    ;SET LAST BIT
        STA NBUFF,X
        INX
        INY         ;FIRST LETTER DONE
        INY         ;SKIP THE DASH
P1      LDA (STL),Y
        STA NBUFF,X
        CMP #32     ;SPACE?
        BEQ P2      ;YES, THEN DONE
        INX
        INY
        CPY #1Ø
        BCC P1
        LDA #32
        STA NBUFF,X ;MAKE LAST A SPACE
P2      STX OFFSET  ;SAVE FOR NEXT TIME
        INC OFFSET  ;INCREMENT OFFSET
        RTS
;
LETTER  .BYTE Ø
;
; FIND FILE NAME IN BUFFER
;
FINDNA  LDX #Ø
L1      LDA NBUFF,X
        INX
        CMP LETTER
        BNE L1
        LDA #68     ;D
        STA FNAME
        LDA #58     ;COLON
        STA FNAME+1
        LDY #Ø
L2      LDA NBUFF,X
        STA FNAME+2,Y
        INX
        INY
        CMP #$2Ø    ;SPACE?
        BNE L2
        DEY
        LDA #46     ;PERIOD
        STA FNAME+2,Y
        INY
        LDA #42     ;ASTERISK
        STA FNAME+2,Y
        INY
        LDA #$2Ø    ;SPACE
        STA FNAME+2,Y
        RTS
;
; GET A LETTER FROM THE KEYBOARD
;
GETLET  LDX #$3Ø    ;IOCB
        LDA #7      ;GET BYTES
        STA ICCMD,X
        LDA #Ø
        STA ICBLL,X
        STA ICBLH,X ;BUFFER LENGTH ZERO
        JSR CIO     ;ACC=LETTER
        CMP #65     ;LESS THAN "A"?
        BCC GETLET
        CMP NUMKEY  ; > HIGHEST LETTER
        BCS GETLET
        ORA #$8Ø    ;SET LAST BIT
        STA LETTER  ;SAVE IT
        RTS
;
; SET RUN ADDRESS TO RTS
;
SETUP   LDA #R&$FF
        STA $Ø2EØ   ;RUN LOW
        LDA #R/256
        STA $Ø2E1
        RTS
;
; BUFFER WITH NAMES WILL BE AT
; END OF PROGRAM.  LETTER WILL
; HAVE LAST BIT SET AND BE
; FOLLOWED BY A HYPHEN.
;
NBUFF   =   *
;
        *=  $Ø2E2   ;INIT ADDRESS
;
        .WORD MENU  ;ADDRESS DATA
;
        .END
```

Is It a Game — Or Is It Real?

NATO COMMANDER



Nato Commander is available for Atari and Commodore 64 computers, 48K disk or cassette. IBM-PC version will be available shortly.

Suggested retail price for this outstanding real-time combat simulation is $34.95.

If you can't find our games at your local store you can order by Master Card, Visa, Money Order, COD or check. Add $2.50 postage and handling. MD residents add 5% sales tax.

- Accelerated real-time simulation
- Full scrolling European Continent
- Demonstration Mode
- Save Game Feature
- Five Battle Scenarios
- Four Skill Levels

PLAYERS HAVE:

- Tactical and Strategic Air Power
- Tactical and Nuclear Forces
- Continuous Battlefield Status and News Service Updates
- Multiple Combat Units, Including: Infantry, Mechanized Armor and Attack Helicopters

MicroProse Software

10616 Beaver Dam Road, Hunt Valley, MD 21030
(301) 667-1151

DEALER INQUIRIES INVITED

ATARI is the registered trademark of Atari, Inc.
Commodore 64 is the registered trademark of Commodore Business Machines, Inc.
IBM-PC is the registered trademark of International Business Machines, Inc.

DISK MISER

A BOOT DISK EMPTY SECTOR USE PROGRAM

16K Disk

by Sait Halman

The Atari boot disk format is extremely inefficient. A single boot-load program ties up the entire disk, even though it may occupy only a few sectors near the beginning.

Disk Miser is a BASIC utility that lets you use the unoccupied sectors on a boot disk to store additional DOS files. It scans the disk to determine which sectors are needed by the boot program, and marks those sectors as "in use" in the disk's Volume Table Of Contents (VTOC). If you aren't familiar with boot disks and the VTOC, refer to Tony Messina's **Disk Tool** articles in **ANALOG** Issues 8 and 9, and Charles Bachand's **Burp!** in Issue 9 for more information. *(Both articles are reprinted in* **The ANALOG Compendium**. *— Ed.).* **Disk Miser** also allows you to back up your boot disks using the "J" option of DOS II. This is considerably faster than using a "dumb" sector copier, because DOS will copy only those sectors which have been marked in the VTOC.

Limitations.

You can use **Disk Miser** with almost any disk whose directory reads "707 FREE SECTORS." An exception should be made in the case of boot programs which load data off the disk during execution, such as text adventures and some multi-level arcade games. Such disks probably have very little empty space anyway.

You should not try to write DOS files onto a disk processed by **Disk Miser**. Regardless of what's in the VTOC, DOS will alter the first three sectors on the disk and wipe out the boot program. Additionally, you should never use the **Miser** on a copy-protected disk, since you might mess up the protection scheme and make the boot program unreadable.

Blank sector handling.

Disk Miser marks in-use sectors in multiples of eight. This simplifies the design of the program and also avoids problems with sectors which must remain blank in order for the boot program to execute (screen data, etc.). A problem may arise if the disk contains eight consecutive blank sectors which occupy the same VTOC byte. I've used **Disk Miser** on over two dozen boot disks and have never encountered this problem; but for safety's sake, I included a manual correction routine in Lines 600-660 of the BASIC program. If the sector map generated by the **Miser** indicates a nonzero (empty) byte surrounded by zero (occupied) bytes, answer the "LOCATION (0 TO EXIT)" prompt with the location of the nonzero byte. In the example shown in **Figure 1**, you would respond to the prompt by typing 13 (RETURN).

(Continued on p. 41.)

```
VOLUME TABLE OF CONTENTS 559

                                  0    0    0    0    0    0
Base location ------------------- 5    0    0    0    0    0
                                  10   0    0    0   255    0
Location 13 (10+3 over)
Manually mark as in use.          15   0    0    0    0    0
                                  20  255  255  255  255  255
                                  25  255  255  255  255  255
                                  30  255  255  255  255  255
VTOC sector (360)                 35  255  255  255  255  255
and Directory                     40  255  255  255  255  255
sectors 361-367
                                  45   0   127  255  255  255
Directory sector
368                               50  255  255  255  255  255
                                  55  255  255  255  255  255
                                  60  255  255  255  255  255
                                  65  255  255  255  255  255
                                  70  255  255  255  255  255
                                  75  255  255  255  255  255
                                  80  255  255  255  255  255
                                  85  255  255  255  255  255

LOCATION (0 TO CONTINUE)?
```

Figure 1.

Program notes.

In Line 110, I initialize the "free sectors" variable FCES to 711 instead of 720 to reflect the 8 sectors (1-8) that are automatically reserved by the Miser, plus the one directory sector (368) which resides in a separate VTOC byte. During execution, sectors 360-367 will be marked as occupied, reducing the maximum number of free sectors to 703.

The POKE 14090,0 in Line 150 marks sectors 1-8 as being "in use." The final POKE statement in Line 210 fools DOS into thinking that the sectors occupied by the boot program are actually a locked, in-use DOS file.

The main loop in Lines 270-310 works as follows. After each block of eight sectors is read into memory, the machine-language subroutine contained in ML$ checks to see if they are blank. If not, a zero is stored in the disk's VTOC to reserve that block. The entire process takes just under four minutes per disk. □

Program variables.

YN$ — For yes/no responses.

A$ — Machine code to get/write sectors:

```
PLA       ; pull # arguments from stack
JSR $E453; use OS to access the disk
RTS       ; back to BASIC
```

DUM$ — Dummy filename.

ML$ — Machine code that checks whether the current 8-sector block is empty (see **Listing 2**).

DCOMND, DBUFLO, DBUFHI, DAUX1, DAUX2 — Memory locations for OS disk handler.

FSEC, FHI, FLO — Number of free sectors.

A — Dummy variable for USR calls.

X,X2 — Dummy variables for FOR/NEXT loops.

SEC, SECHI, SECLO — Current sector number.

BUF, BUFHI, BUFLO — Current buffer address.

USEC, UHI, ULO — Number of used sectors.

Listing 1.
BASIC Program.

```
10 REM * DISK MISER
20 REM * by Sait Halman
30 REM * ANALOG Computing #17
40 REM
50 REM * ONE-TIME INITIALIZATION
60 DIM YN$(3),A$(5),DUM$(8),ML$(36)
70 POKE 769,1:DCOMND=770:DBUFLO=772:DB
UFHI=773:DAUX1=778:DAUX2=779
80 FOR X=1 TO 5:READ A:A$(X)=CHR$(A):N
EXT X:FOR X=1 TO 36:READ A:ML$(X)=CHR$
(A):NEXT X
90 REM
100 REM * PER DISK INITIALIZATION
110 FSEC=711:POKE DCOMND,82
120 ? "↖INSERT DISK":? "DUMMY FILENAME
 (8 CHARS)";:INPUT DUM$:IF LEN(DUM$)=0
 THEN DUM$="BOOTDISK"
130 REM
140 REM * GET SECTORS 360-361
150 POKE DAUX1,104:POKE DAUX2,1:POKE D
BUFLO,0:POKE DBUFHI,55:A=USR(ADR(A$)):
POKE 14090,0
160 POKE DAUX1,105:POKE DAUX2,1:POKE D
BUFLO,128:POKE DBUFHI,55
170 A=USR(ADR(A$))
180 REM
190 REM * ADD EXTENDER TO FILENAME
200 REM * AND PASS TO DIRECTORY
210 FOR X=1 TO 8:POKE 14212+X,32:NEXT
X:POKE 14208,106
220 FOR X=1 TO LEN(DUM$):POKE 14212+X,
ASC(DUM$(X,X)):NEXT X
230 DUM$="DUM":FOR X=1 TO LEN(DUM$):PO
KE 14220+X,ASC(DUM$(X,X)):NEXT X
240 REM
250 REM * GET 8 SECTORS AT A TIME
260 REM * AND CHECK FOR BLANKS
270 FOR X=1 TO 89:FOR X2=0 TO 7
280 SEC=8*X+X2:SECHI=INT(SEC/256):SECL
O=SEC-256*SECHI:POKE DAUX1,SECLO:POKE
DAUX2,SECHI
290 BUF=14336+128*X2:BUFHI=INT(BUF/256
):BUFLO=BUF-256*BUFHI:POKE DBUFLO,BUFL
O:POKE DBUFHI,BUFHI
300 A=USR(ADR(A$)):NEXT X2:POKE 14079,
X:IF USR(ADR(ML$))<>255 THEN POKE 1409
0+X,0:FSEC=FSEC-8
310 NEXT X
320 REM
330 REM * MANUAL CORRECTIONS
340 ? "↖  VOLUME TABLE OF CONTENTS  ";
FSEC:FOR X=0 TO 17:POSITION 2,X+2:? 5*
X:FOR X2=0 TO 4
350 POSITION 6*X2+10,X+2:? PEEK(14090+
5*X+X2);:NEXT X2:NEXT X
360 POSITION 2,22:? "LOCATION (0 TO CO
NTINUE)";:INPUT X:IF X=0 THEN 420
370 IF PEEK(14090+X) THEN POKE 14090+X
,0:FSEC=FSEC-8
380 GOTO 340
390 REM
400 REM * FIND # FREE/USED SECTORS
410 REM * AND PASS TO DIRECTORY
420 FHI=INT(FSEC/256):FLO=FSEC-FHI*256
:USEC=720-FSEC:UHI=INT(USEC/256):ULO=U
SEC-UHI*256
430 POKE 14083,FLO:POKE 14084,FHI
440 POKE 14209,ULO:POKE 14210,UHI
450 REM
460 REM * WRITE SECTORS 360-361
470 POKE DCOMND,87:POKE DAUX1,104:POKE
 DAUX2,1:POKE DBUFLO,0:POKE DBUFHI,55:
A=USR(ADR(A$))
480 POKE DAUX1,105:POKE DAUX2,1:POKE D
BUFLO,128:POKE DBUFHI,55:A=USR(ADR(A$)
)
```

```
490 REM
500 REM * ALL DONE; NEED ANOTHER?
510 ? "↖FREE SECTORS:";FSEC
520 ? "USED SECTORS:";USEC
530 ? "↓ANOTHER DISK";:INPUT YN$:IF YN
$(1,1)="Y" THEN 110
540 END
550 REM
560 REM * MACHINE CODE FOR A$
570 DATA 104,32,83,228,96
580 REM
590 REM * MACHINE CODE FOR ML$
600 DATA 104,162,0,134,213,189,0,56,20
8,21,189,0,57,208,16,189,0,58
610 DATA 208,11,189,0,59,208,6,232,208
,233,202,208,2,162,0,134,212,96
```

•

CHECKSUM DATA
(see p. 15)

```
10 DATA 798,805,296,257,589,924,582,88
0,267,625,668,849,83,539,458,8620
160 DATA 85,370,98,684,378,661,489,982
,88,416,986,240,806,619,634,7536
310 DATA 765,84,126,266,87,369,743,727
,105,746,385,529,312,355,95,5694
460 DATA 323,798,937,107,362,122,76,23
4,46,97,737,101,106,81,196,4323
610 DATA 229,229
```

•

Listing 2.
Assembly source for ML$.

```
; DISK MISER BY SAIT HALMAN
; ASSEMBLY SOURCE FOR ML$
;
FR0 =   $D4     ; BASIC RETURN REGISTER
BUF1 =  $3800   ; SECTOR 1/2 BUFFER
BUF2 =  $3900   ; SECTOR 3/4 BUFFER
BUF3 =  $3A00   ; SECTOR 5/6 BUFFER
BUF4 =  $3B00   ; SECTOR 7/8 BUFFER
;
    *=  $3600   ; COULD GO ANYWHERE
;
    PLA         ; # USR PARAMETERS; IGNORE
    LDX #0      ; ZERO MSB OF RESULT
    STX FR0+1   ; AND INIT LOOP
LOOP
    LDA BUF1,X  ; IF BYTE IS NZ,
    BNE DATA    ; MARK BLOCK
    LDA BUF2,X  ; CHECK 2ND PAGE
    BNE DATA
    LDA BUF3,X  ; 3RD PAGE
    BNE DATA
    LDA BUF4,X  ; 4TH PAGE
    BNE DATA
    INX         ; CONTINUE LOOP
    BNE LOOP    ; UNTIL DONE
;
; ALL SECTORS EMPTY, SO
; TELL BASIC WITH A "TRUE" FLAG
;
    DEX         ; X = $FF
    BNE EXIT
;
; DATA BYTE DISCOVERED, SO
; PASS A ZERO BACK TO BASIC
;
DATA
    LDX #0
EXIT
    STX FR0
    RTS         ; RETURN TO BASIC
;
    .END
```

OPERATION WHIRLWIND
by Roger Damon
BRODERBUND SOFTWARE
1938 Fourth Street
San Rafael, California 94901
48K Disk $39.95

by Pat Kelley

Almost every Atari game released these days is either a retread of an arcade hit or a pale imitation of a past home computer success. Refreshing ideas are few and far between. So it was with some trepidation that I tore the shrinkwrap off of **Operation Whirlwind**, the latest strategy game from Broderbund. It looked like little more than a juiced-up version of Chris Crawford's **Eastern Front 1941** in glossy packaging. After all, the scenario is similar: You command an attack force comprised of Infantry, Armor (heavy and light), Artillery and Engineer/Shock Troops. Your objective is to engage and bypass enemy units of similar strength, and to occupy cities with minimum loss to your forces. This, I was happy to discover, is where all similarity ends.

The game.

Whirlwind begins with your forces placed on the far left side of a scrolling relief map, with each unit of your army designated by a symbol. Positioning your forces is a simple matter; simply place a joystick-controlled cursor over the desired unit, and move at will. Holding down the fire button calls up a text window full of information about the unit (class, strength, mobility and firepower). This window will also inform you when you cannot move a certain unit into an area, whether a unit is under fire, when a unit has overextended itself or if you're attempting to move one unit through another—a definite no-no!

The joystick lets you fine-scroll across the map and view the obstacles that await you. The map represents a total area of fifteen kilometers from end to end, a distance you must blitz across if you are to capture the enemy city and win victory for your Fatherland. But the road to Valhalla is a rough one, indeed.

Blazing combat.

Once you activate your forces, **Operation Whirlwind** is underway, and attack can come at any time. Cautious movement at this stage is advised. As in real combat, you are advancing into unfamiliar territory, and you never know exactly where your adversary is until he gives himself away. Enemy units are invisible until they open fire on you, usually at whites-of-your-eyes range.

Commands to your units at this stage are given in five distinct phases:

1. **Command.** In this stage of the conflict, you must determine the status of each unit in your command. You decide which units will *Dig In* and hold their ground, and which will *Become Combat Ready.* The Command phase allows you to replenish a unit's strength if damaged under fire. *Dig In* gives weary units a chance to recuperate, and hopefully fight another day. I'm personally something of a tyrant when it comes to this, as all of my units are on the advance all of the time. George Patton would be proud. . .

2. **Movement.** This is where the stuff really starts to fly. **Whirlwind**'s Movement phase gives you an opportunity to carefully feel your enemy out. Select units are commanded to slowly advance in an attempt to draw enemy fire, thereby pinpointing their heretofore secret positions. Only your movement will persuade them to come out and reveal themselves.

Operation Whirlwind.

3. **Combat.** The enemy has shown himself at last, and the real confrontation begins. This is the phase that separates the men from the boys.

Your best combat tool is your artillery. Blessed with withering firepower and virtually unlimited range, your big guns can mortally wound the enemy. The computer displays a crosshair for you to place over the enemy unit you wish to engage, and the rest is up to you. By pressing joystick trigger, you order the selected unit to open fire and send out the hurt. How much hurt you dispense is for you to decide. You'll learn quickly which of your units are the most powerful, and which the least. The devastating power of your artillery is a joy to behold, and will get you out of many a tight spot.

Once you feel that you've gained enough headway in a particular firefight, or have decimated enough enemy units, it's time to move on to the next phase:

4. **Assault Orders.** Now you can finally get some use out of those Engineers who've been hitching a free ride. Use the joystick to select which of the Engineer or Infantry units you wish to begin the assault, and send them on their way.

5. **Assault.** This is when you enjoy the fruit of your labors. Blown bridges can be rebuilt now if you have ordered Engineers to do so in the previous phase, paving the way for your armored units to roar into the city. Infantry troops begin shock action, mopping-up and dislodging any remaining bastions of the enemy overrun in your blitzkrieg. **Whirlwind**'s Assault Phase lets you watch routed enemy units pulling back for a last-ditch defense of their citadel; a heartwarming sight. Don't get too optimistic at this point, though. The battle has not yet been won.

As your remaining forces group for their final rush into the city, the first in a long series of house-to-house, street-to-street exchanges begin. As before, enemy units remain hidden until you practically step on their toes, but this time you'll notice a dogged sense of defiance in the way they appear, seemingly in the jaws of defeat, to fight on. Make no mistake: this final push into the city is no cakewalk, and — more often than not — you'll want to pull your hair out as you try to stamp out pocket after pocket of resistance before **Whirlwind** makes its decision on your future as a career officer.

Promotion... or the Eastern Front?

The computer is a stern and very particular judge. Once it has bestowed a performance rating upon you at game's end, no amount of screaming, cajoling, abuse or keypounding will sway it. The only thing that can help you win a more favorable rating is you, and your previous experience as a leader of men.

These are the possible ratings or "victory levels" awarded at the end of a round of **Operation Whirlwind**:

1. **Questionable.** Your victory is tenuous at best. An enemy counterattack would probably dislodge you, and as such you'd have to trade your Officer's uniform in for a burlap one with numbers stenciled on the side. Hang your head in shame.

2. **Marginal.** You've clawed a foothold into enemy territory, and are holding on by the skin of your teeth. A few more casualties and you'd be going by the name of **Questionable**. Next time, try to be more forceful in your assault. Go read *The Tank In Attack* by Irwin Rommel.

3. **Tactical.** The operation has succeeded totally. You have dealt a severe blow to enemy morale, and have crushed their will to fight. In securing the city, you have won a major victory, and have provided Military High Command with another steppingstone in the war. Congratulate yourself on a job well done.

4. **Strategic.** Your swift advance and lightning assault has created a New Front in your sector, and you have practically assured victory for all future exercises within the region. This campaign has bolstered your career as an officer, and has made you the one to watch in the future. Add another Cross of Iron to your collection.

5. **Breakthrough.** Your victory has become a turning point in the war. With the citadel's capture comes the assurance that the war will soon be over, and unconditional victory is yours. Nice shooting, General.

Well, there you have it. If you thrive on the science of war, Roger Damon's **Operation Whirlwind** is definitely your cup of tea. This fine simulation proves that computer games don't have to be part of the crowd to succeed. □

Ask Sally Forth

by Sally Forth

I want to thank you for your article in Issue #13 of **ANALOG**. *It was a great help in learning FORTH. I was able to compare the BASIC and FORTH programs, and finally understood a couple of simple techniques.*

I've been playing with **valFORTH** *for almost six months with very little success. True, I haven't devoted all of my energies to learning FORTH, but I do have about ten different books on the language. They all fail in one thing: How to actually program something. They go to great pains to explain the stack, numbers, and why the language is great. But examples of code and explanations of the how and whys, they do not have.*

I have a question that you might be able to address in one of your articles. I read keycodes a lot in my BASIC programs, but I have yet to figure out what I have to do to perform the same function in FORTH. My BASIC code looks something like this:

```
10 KEYBOARD=53775:UNTOUCH=255:RESPONSE
=764
300 KEY=PEEK(KEYBOARD):IF KEY=UNTOUCH
THEN 300
310 IF PEEK(RESPONSE)=15 THEN CN=CN+1:
GOTO 490
320 IF PEEK(RESPONSE)=14 THEN CN=CN-1:
GOTO 490
330 IF PEEK(RESPONSE)=12 THEN 520
340 GOTO 300
```

My main hangup is figuring out which control structure to use (IF/ELSE, BEGIN/UNTIL or WHILE/REPEAT) to accomplish the job of Line 300. When I use the debugger and watch the stack, the values always seem to be wrong.

I appreciate any help you would be willing to provide. Please keep the articles coming. There's a lot of interest in my local user group in using FORTH, but it seems to be too difficult to learn.

Jim Watson
Corpus Christi, Texas

Don't give up hope! Your problem is a bad case of BASIC On The Brain, not incurable, but difficult to shake off.

Your BASIC code for reading the keyboard is more complicated than it has to be. You could do the same thing with:

```
10 KEYCODE=764:CLEAR=255
300 K=PEEK(KEYCODE):IF K=CLEAR THEN 30
0
310 POKE KEYCODE,CLEAR:IF K=15 THEN CN
=CN+1:GOTO 490
320 IF K=14 THEN CN=CN-1:GOTO 490
330 IF K=12 THEN 520
340 GOTO 300
```

The FORTH equivalent looks like this:

```
( First, declare the variables
( K and CN. )

0 VARIABLE K  ( holds last keypress )
0 VARIABLE CN ( program variable    )

( Now declare KEYCODE and RESET as
( FORTH constants. )
```

```
764 CONSTANT KEYCODE
255 CONSTANT RESET

( The following definition is used to
( scan the hardware's keyboard
( register until a key is pressed.
( The register is then reset and the
( internal keycode is left on the
( stack. )

: GETKEY ( --- n )

  BEGIN
    KEYCODE C@ RESET <>
  UNTIL

  KEYCODE C@ RESET KEYCODE C! ;


( Let's define a couple of dummy
( words [LINE490 and LINE520]
( to represent the corresponding
( lines of BASIC code.  You would
( replace these with definitions that
( suit your application. )

: LINE490
  ." LINE 490 "   ( display line # )
  ." CN = "       ( and value of   )
  CN @ . CR ;     ( variable CN    )

: LINE520
  ." LINE 520 "   ( same as LINE490 )
  ." CN = "
  CN @ . CR ;


: DOKEY ( the control selector )

  BEGIN ( start indefinite loop )

    GETKEY   ( fetch keypress and )
    K !      ( save it in K       )

    K @ 15 =  ( IF K=15 )
      IF

      1 CN +! ( CN=CN+1            )
      LINE490 ( execute LINE490    )
      1       ( leave a "true" flag )
              ( on stack to exit    )
              ( BEGIN/UNTIL loop    )

      ELSE    ( otherwise ... )

        K @ 14 = ( IF K=14 )
          IF

          -1 CN +! ( CN=CN-1     )
          LINE490  ( do LINE490  )
          1        ( and exit loop )

          ELSE

            K @ 12 = ( IF K=12 )
              IF

              LINE520 ( do LINE520 )
              1       ( and exit   )

              ELSE ( if all else   )
                   ( fails,        )
              0    ( tell UNTIL to )
                   ( keep looping  )

              ENDIF

          ENDIF

      ENDIF

  UNTIL  ( end indefinite loop )
  ;      ( end definition      )
```

I assume that your prodigious FORTH library includes Leo Brodie's *Starting FORTH* (Prentice-Hall). Concentrate your reading on this book; it's the clearest introduction to the FORTH language ever published.

Did I do that?

I goofed. Last month's **FPLOT** utility had a couple of itsy-bitsy bugs, which I didn't notice until it was too late to fix them in the magazine.

The ROWCRS and COLCRS constants declared in screen #1 refer to the wrong locations. Even more embarrassing, my plot-mask tables PMASKS and NMASKS were written backwards! **FPLOT** still works after a fashion, but the horizontal positioning of the pixels is somewhat skewed; and you can't follow up an FPLOT with an OS DRAWTO with reliable results. Blame it on a Christmas bottle of Bailey's Irish Cream, which makes even reverse Polish notation look sensible in holiday doses.

The remedy is easy and effective. Simply replace screens 1-3 of the original **FPLOT** listing with the following:

(Forth screens next page.)

```
SCREEN #1

( HIGH SPEED MODE F PLOTTER )

( Reserve space for tables )

DECIMAL

LABEL YLOWS 192 ALLOT  ( lsbs )
LABEL YHIGHS 192 ALLOT ( msbs )

( OS equates )

90 CONSTANT ROWCRS ( REVISED )
91 CONSTANT COLCRS ( REVISED )
88 CONSTANT SAVMSC

                                -->
SCREEN #2

( HIGH SPEED MODE F PLOTTER )

2 BASE ! ( for convenience )

LABEL NMASKS ( REVISED )

     01111111 C,  01111111 C,
     10111111 C,  10111111 C,
     11011111 C,  11011111 C,
     11101111 C,  11101111 C,
     11110111 C,  11110111 C,
     11111011 C,  11111011 C,
     11111101 C,  11111101 C,
     11111110 C,  11111110 C,

                                -->
SCREEN #3

( HIGH SPEED MODE F PLOTTER )

LABEL PMASKS ( REVISED )

         0 C,  10000000 C,
         0 C,   1000000 C,
         0 C,    100000 C,
         0 C,     10000 C,
         0 C,      1000 C,
         0 C,       100 C,
         0 C,        10 C,
         0 C,         1 C,

DECIMAL

                                -->
```

Now that **FPLOT** is fully operational, we can move on to more advanced graphics concepts like line drawing. Screens 18-31 contain the **valFORTH assembly definition of VFAST, which draws vertical lines in ANTIC mode F at breathtaking speed. It's** great for filling in blocks of screen, drawing bar charts and various types of animation. And it works. I promise!

VFAST and the demos that follow it share many of the FORTH words we defined last month for **FPLOT**. That means you have to compile the **FPLOT** screens into your system *first*, before loading **VFAST**. Here's the procedure for LOADing and testing **VFAST**:

1. Replace screens 1-3 of **FPLOT** with the new code shown above.
2. Make sure the **valFORTH** Assembler, Graphics and Color libraries are compiled into your system. If not, LOAD them in now or you'll wish you did. (Check Valpar's documentation for more information.)
3. Now LOAD the complete set of **FPLOT** screens (1-17) into your system. Track down compilation errors and re-LOAD as needed until everything's okay.
4. Now you can LOAD screens 18-31. Small typos in **FPLOT** may now become evident. If not, answer the "ok" prompt with **SLOWLINE** and watch how long it takes Atari's clunky old PLOT and DRAWTO routines to fill a mode F screen with 320 vertical lines. Then try **FASTLINE** and all your effort will seem worthwhile. I got execution times of 2602 jiffies for **SLOWLINE** and just 198 jiffies for **FASTLINE**, over thirteen times faster!

"We control the vertical."

The rules for using **VFAST** are similar to those for **FPLOT. First, set up the high-res screen with a 24 GR. call. Next, execute the word PLOTSETUP** to initialize the look-up tables that make **FPLOT** and **VFAST** so swift. Don't forget to call **PLOTSETUP** or your machine will lock up the instant you call **VFAST**! Select a drawing color with **n COLOR, where n equals 1 for foreground or 0 for back**ground. Both **FPLOT** and **VFAST** are now ready to use. The syntax of **VFAST** is:

```
X1 Y1 Y2 ---
```

where **X1** is the X-position of the desired vertical line, **Y1** is the Y-position of the first endpoint you **want to draw and Y2** the position of the other endpoint. Thus the command:

```
20 20 50 VFAST
```

will produce exactly the same result onscreen as:

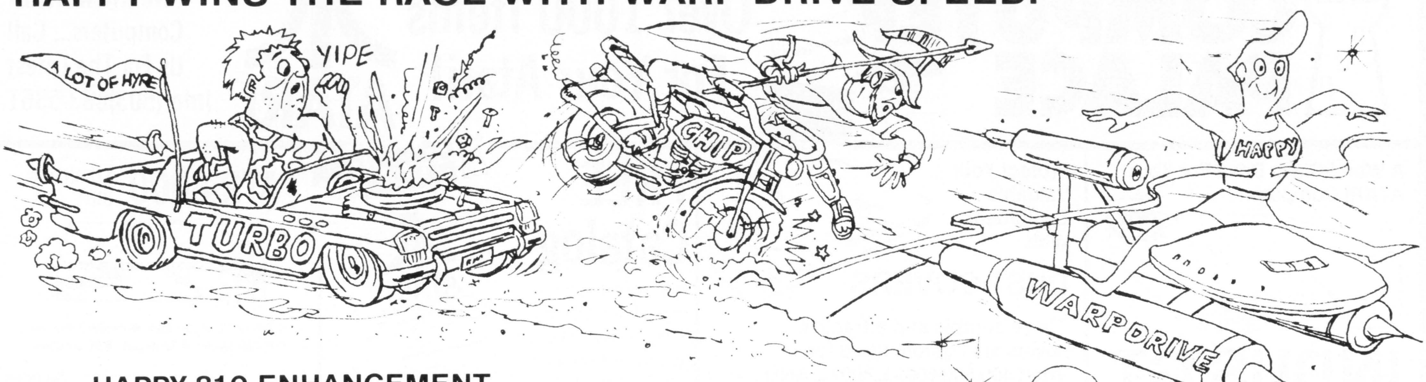
```
20 20 PLOT ( or FPLOT )
20 50 DR.
```

except that **VFAST** will finish drawing a lot sooner.

VFAST is friendlier than **FPLOT** because it checks to make sure your parameters are within legal ranges. Pass **VFAST** a bad X1, Y1 or Y2 value and it will jump back into FORTH without drawing anything. Venturesome programmers may wish to modify **VFAST** so that it "clips" lines at the edge of the screen. Incidentally, **VFAST** leaves the coordinates of the last point it has drawn (X1, Y2) in the Atari's OLDROW **and** OLDCOL **registers, so you can** follow it up with a regular DRAWTO command if you like and the operating system will never know the difference. I know I said that last month too, but this time I really mean it!

That's all there is to lightning-fast vertical lines in **valFORTH**. Bet you can't guess what I'll be showing you next month. □

HAPPY WINS THE RACE WITH WARP DRIVE SPEED!



HAPPY 810 ENHANCEMENT

- The only change needed to run all WARP DRIVE software
- Plug in P.C. board requires no permanent modifications
- Proven reliable in thousands of installations, reduces disk drive wear
- Comes completely assembled and tested, just plug in and use
- Full one year parts and labor guarantee, compatible with existing software
- High quality printed circuit board with gold connectors

NO ONE ELSE HAS THIS PERFORMANCE
Unenhanced whole disk (ATARI rev B format) read time: 112 seconds
Unenhanced whole disk (ATARI rev C fast format) read time: 89 seconds
ENHANCED 810 whole disk (any format) read time with standard software: 68 seconds
ENHANCED 810 whole disk (any format) read time with WARP DRIVE software: 43 seconds
Standard software whole disk write and verify time: 238 seconds
WARP DRIVE software whole disk write and verify time: 62 seconds

NEW HAPPY WARP DRIVE SOFTWARE

WARP SPEED HAPPY BACKUP PROGRAM
- Completely automatic: nothing to figure out, insert disks and press return
- Only program on the market guaranteed to backup any disk
- Can write to a blank disk: format write and verify in one operation
- Automatic program tracing: copies only the tracks that are used
- Efficient memory utilization: reduces the number of disk insertions
- Requires only one ENHANCED disk drive, backups will work on a standard drive

WARP SPEED MULTI DRIVE HAPPY BACKUP PROGRAM
- Same features as above plus support of multiple ENHANCED drives
- Can be used with up to 4 ENHANCED drives
- Source and all destination drives read and write in parallel
- Format write and verify 3 complete disks in less than 3 minutes

WARP SPEED HAPPY COMPACTOR PROGRAM
- Reduces the number of disks required to backup your library
- Combines up to 8 self booting disks into 1 disk with a menu
- Compacted disks run only on an ENHANCED drive
- Pays for itself by saving on disks
- Single or dual ENHANCED drive operation

HAPPY WARP DRIVE DOS
- Improves ATARI DOS 2.0S to use warp speed reading and write with verify
- Use all features of BASIC, PILOT, FMS, and DUP at top warp speed
- Warp speed I/O software module available separate from DOS

HAPPY WARP DRIVE SECTOR COPY PROGRAM
- Standard format whole disk read, write and verify in 105 seconds
- Use with sngle or dual drives, mix ENHANCED and NON-ENHANCED drives

HAPPY CUSTOMIZER PROGRAM (sold separately $99.95)
- Creates custom format disks of any specification
- Any type bad sector, duplicate sector numbers, or interleave
- Easy to use but requires an advanced level user to interpret the results

REVIEWED IN POPULAR MAGAZINES

A.N.A.L.O.G. COMPUTING—July/August 1983 *"...The installation instructions for the Happy 810 Enhancement are among the best I have ever seen. ...The Happy 810 Enhancement is one of the most powerful hardware modifications available to ATARI computer owners."*

ANTIC—July 1983 *"The difference between a normal ATARI 810 disk drive and one equipped with Happy is like the contrast between mass transit and the automobile. A car costs you more initially, but improves the quality of your life. Similarly, if you use your disk drive a lot, installing Happy will markedly enhance your programming life."*

SPECIAL SUGGESTED RETAIL PRICE BEFORE FEBRUARY 28, 1984: *Get the HAPPY 810 ENHANCEMENT with the single and multi drive HAPPY BACKUP PROGRAM, plus the HAPPY COMPACTOR PROGRAM, plus the HAPPY DRIVE DOS, plus the HAPPY SECTOR COPY, all with WARP DRIVE speed, including our diagnostic for $249.95. Existing registered ENHANCEMENT owners may upgrade to WARP DRIVE speed for $15.00 with no hardware changes.!*

Price includes shipping by air mail to U.S.A. and Canada. Foreign orders add $10.00 and send an international money order payable through a U.S.A. bank. California orders add $16.25 state sales tax. Cashiers check or money order for immediate shipment from stock. Personal checks require 2-3 weeks to clear. Cash COD available by phone order and charges will be added. No credit card orders accepted. ENHANCEMENTS for other ATARI compatible drives coming soon, call for information. Please specify -H model for all drives purchased new after February 1982, call for help in ENHANCEMENT model selection. Dealers now throughout the world, call for the number of the dealer closest to you.

HAPPY COMPUTERS, INC. • P. O. Box 1268 • Morgan Hill, California 95037 • (408) 779-3830

SCREEN #18

```
( VFAST Vertical Line Drawing  )
( REMEMBER: You MUST have the
( FORTH screens from Issue #16
( [pp. 81-83] compiled into
( your system BEFORE you LOAD
( these screens!  And don't
( forget to repair screens 1-3
( as shown in this month's
( column, either. )
ASSEMBLER
        112 CONSTANT MASKB
        113 CONSTANT LEFT
        114 CONSTANT VECT
        115 CONSTANT SADDR     -->
```

SCREEN #19

```
( VFAST Vertical Line Drawing  )
( This is the assembly code for
( VFAST.  Syntax is:
(          X1 Y1 Y2 ---
( WHERE:
( X1 is the fixed horizontal
( position of the line [0-319],
( Y1 is the Y-coordinate of
( the first end of the line you
( want to draw [0-191], and
( Y2 is the Y-coordinate of the
  other end [0-191]. )
                               -->
```

SCREEN #20

```
( VFAST Vertical Line Drawing  )
( Line color is controlled by
( the standard valFORTH word
( COLOR.  Legal COLOR values
( are 0 and 1.  See valFORTH's
( documentation for more info. )
CODE VFAST
      # 3 LDA, ( # DROP values )
    SETUP JSR, ( move into N   )
    XSAVE STX, ( preserve X    )
          CLD, ( for safety    )
                               -->
```

SCREEN #21

```
( VFAST Vertical Line Drawing  )
      PNTR LDA, ( get lsb of X1 )
   PNTR 1+ LDY, ( and msb       )
      # 1 CPY,  ( if X1 is      )
                ( greater than  )
CS IF,
     # 64 CMP,  ( 320, abort    )
                ( the VFAST and )
  CS IF,        ( return to     )
     NEXT JMP,  ( FORTH         )
  ENDIF,
ENDIF,
                               -->
```

SCREEN #22

```
( VFAST Vertical Line Drawing  )
   COLCRS STA, ( save X1 for   )
COLCRS 1+ STY, ( the OpSys     )
      XLO LDA, ( if Y1 is      )
  # 192 CMP,   ( greater than  )
               ( 192, abort    )
CS IF,
     NEXT JMP, ( the VFAST     )
ENDIF,
        N LDA, ( also be sure  )
    # 192 CMP, ( Y2 is within  )
                               -->
```

SCREEN #23

```
( VFAST Vertical Line Drawing  )
CS IF,         ( range; else   )
     NEXT JMP, ( abort VFAST   )
ENDIF,
      XLO CMP, ( is Y1 bigger  )
               ( than Y2?      )
CS IF,         ( if so, then   )
               ( calc the      )
      XLO SBC, ( difference &  )
      # 1 LDY, ( set drawing   )
               ( direction to  )
               ( "down" [1]    )
                               -->
```

SCREEN #24

```
( VFAST Vertical Line Drawing  )
ELSE,          ( if Y2 is      )
               ( larger, then  )
      XLO LDA, ( calc the      )
          SEC, ( difference &  )
        N SBC, ( set drawing   )
    # 255 LDY, ( direction to  )
               ( "up" [-1]     )
ENDIF,
               ( save the      )
     LEFT STA, ( delta and the )
     VECT STY, ( direction for )
               ( later use     )
                               -->
```

SCREEN #25

```
( VFAST Vertical Line Drawing  )
     PNTR LDA, ( fetch X1 and  )
  PNTR 1+ LSR, ( divide it by  )
       .A ROR, ( 8 for use as  )
       .A LSR, ( a Y-index     )
       .A LSR, ( into the      )
          TAY, ( mode line     )
     PNTR LDA, ( now mask X1   )
      # 7 AND, ( to get bit    )
       .A ASL, ( position,     )
   CLRBYT ORA, ( superimpose   )
               ( COLOR data &  )
    MASKB STA, ( save result   )
                               -->
```

SCREEN #26

```
( VFAST Vertical Line Drawing )
      XLO LDX, ( init Y-coord )
   ROWCRS STX, ( for drawing )
( Here begins the loop that
( actually draws the line for
( you. )
BEGIN,
 YLOWS ,X LDA, ( fetch addr of )
    SADDR STA, ( first byte in )
YHIGHS ,X LDA, ( the mode line )
 SADDR 1+ STA,
                              -->
```

SCREEN #27

```
( VFAST Vertical Line Drawing )
    MASKB LDX, ( retrieve the )
               ( color mask )
 SADDR )Y LDA, ( fetch the )
               ( proper screen )
               ( byte, )
NMASKS ,X AND, ( zero out the )
               ( plot bit, )
PMASKS ,X ORA, ( superimpose )
               ( color data & )
 SADDR )Y STA, ( proudly show )
               ( the byte )
                              -->
```

SCREEN #28

```
( VFAST Vertical Line Drawing )
   ROWCRS LDA, ( update the )
          CLC, ( Y-coord for )
     VECT ADC, ( the next plot )
   ROWCRS STA, ( restore table )
          TAX, ( index )
     LEFT DEC, ( keep drawing )
               ( till no more )
EQ UNTIL,      ( Y-coords left )
    XSAVE LDX, ( restore X and )
     NEXT JMP, ( we're done! )
            C;                -->
```

SCREEN #29

```
( VFAST Demonstration Words )
( You remembered to compile
( last month's screens, didn't
( you? Of course you did. )
: LINEINIT
  24 GR.     ( graphics mode )
  14 709 !   ( init screen & )
  1 COLOR    ( line colors )
  0 TOCK !   ( reset timers )
  ;
                              -->
```

SCREEN #30

```
( VFAST Demonstration Words )
: SLOWLINE ( using Atari's )
           ( CIO PLOT and DRAW )
  LINEINIT
  320 0 DO
    I 0 PLOT I 191 DR.
  LOOP
  SHOWTIME ." CIO lines."
  CR ;
                              -->
```

SCREEN #31

```
( VFAST Demonstration Words )
: FASTLINE   ( using our new )
             ( word VFAST )
  LINEINIT
  PLOTSETUP ( don't forget! )
  320 0 DO
    I 0 191 VFAST ( simple, )
  LOOP            ( eh? )
  SHOWTIME ." VFAST."
  CR ;
```

LODE RUNNER
by Doug Smith
BRODERBUND SOFTWARE
17 Paul Drive
San Rafael, California 94903
48K Disk $34.95

by Lee Pappas

Here is proof that a video game can be entertaining without laser beams, bombs or explosions.

Lode Runner centers around a highly skilled Galactic Commando, who is placed deep inside the treasure-filled catacombs of the evil Bungeling Empire. Your mission: Grab the gold. Each of the 150 (yes, 150) levels contains several kegs of gold, which you must approach and recover. But Guards of Stupidity are patrolling these chambers, and they'll do everything they can to try and catch you.

Each level is made up of brick walkways, ladders, hand over hand bars, gold, guards and you. You have the ability to run, jump down, climb, drill holes and die. The guards can do all of these, except drill and die.

What's going on.

A level may contain from two to five guards. They'll jump, climb and run to either where you are, or where they think you are going. It's not that they're particularly bright; I think it's blind luck. You'll catch them doing some funny things, sometimes resembling the Keystone Kops.

You appear as a little-bigger-than-a-stick figure standing on a blue platform. The platforms can be composed of brick, which you can drill through, or solid rock, which is impervious to your laser drill. Trapping the guards is usually accomplished by drilling a pit and forcing the guards to fall in. After a few seconds, though, the guard can climb out, so you must either time your moves so he gets buried (pits fill in after 10 seconds), or dig several pits.

Anytime a guard gets buried, another will take his place, usually appearing near the top of the screen. This can work against you, but if you're a skilled player you'll often be grateful for it. If you manage to get close to level 30, you'll see gold you can't reach or climb to by yourself. By letting the guards pick up the gold (they do this quite a bit), they may bring it to a point accessible to you. You can steal the gold from a guard by making him fall into a pit, then walking on his head until he gives up the loot.

To make things even trickier for you and the guards, trap doors will drop anyone who walks over them, and undiggable floors may be the death of you, offering no protection from guards on the same level. Only when you retrieve all of the gold on a level (including what the guards are carrying) will an escape ladder appear for your getaway.

Options.

Lode Runner gives you a choice of either keyboard or (preferably) joystick control. The ESC key will pause/resume the game whenever you need a break. Occasionally, you may get trapped or stuck with no way out. When that happens, pressing CTRL — "A" will abort the round and start you back at the beginning, but you'll forfeit a life. The left and right cursor keys slow down or speed up the game speed respectively, and CTRL — "D" actuates forward or backward drilling.

My favorite buttons are CTRL keys "U" and "F." These allow you to cheat by adding extra lives (up to 999) and allowing you to proceed manually through the levels without having to play each one through, although the computer still has to load each screen off the disk in sequence. The price for this immorality is the inability to record your high-score on the disk.

The sound effects, animation, color and graphics in **Lode Runner** are only marginal, but I'm not too upset about it. I'm fed up with all this talk of hi-res graphics and bla-bla-bla. **Lode Runner** is *game play*, with graphics and sound to support it. This is a welcome relief from "games" like **Astro Chase**, which are little more than graphics showcases.

Lode Runner is more than just action, though. Strategy and tactics are all-important, and there'll be times when thought alone will get you that last keg of gold or save your hide.

By now, it's obvious I find little fault with the game, but the next feature is strictly frosting on the cake.

If you don't like it...

After 150 screens, I can't imagine anyone getting bored with **Lode Runner**, but let's suppose for the sake of argument that the game is too tough or too easy, or you're tired of cheating. A well-designed option allows you to literally create your own **Lode Runner** screens and save them on disk.

Lode Runner's screen editor lets you enter in the level number you want it to be, how many guards will chase you, and even where the escape ladders should go. Gold, brick, trapdoors, hand bars and everything else can be placed by you. After saving your new design on disk, you can immediately play the screen to see if it's OK, or recall it for further editing. Screens can be created and played in just minutes, and the game automatically handles all of the logistics of guys running around, falling and dying.

Even without this incredible option, **Lode Runner** would be another real winner from Broderbund. Author Doug Smith should be congratulated for coming up with a terrific piece of entertainment software. ☐

HOW TO LIVE WITHOUT DOS

16K Disk

by Dan Higgins

Anyone with an Atari disk drive probably uses a disk operating system (such as Atari DOS 2.0S) for communicating with the drive and controlling disk operation. Did you know, however, that you don't really need DOS to input or output information to or from your disk? This article describes a method for disk input/output that is independent of DOS. Don't throw away your DOS disks, however! DOS is still a very useful tool, even if not an absolute necessity.

First consider a typical single-density Atari disk. When the disk is formatted, it has 720 data sectors available, each of which can hold 128 bytes of information. Multiplying the number of sectors by the number of bytes per sector gives a total of 92,160 bytes per disk. Atari DOS uses some of this space for its own purposes, however, so that the user can store only about 88,000 bytes of data with DOS. For example, sector 360 (decimal) is used by DOS to store the Volume Table of Contents (VTOC). This VTOC sector contains a bit map which DOS uses to keep track of which sectors are in use.

Sectors 361 to 368 contain the DOS disk directory, which keeps a record of what files are on the disk. Also, DOS 2.0S has a bug in it that makes it unable to access sector 720. There are thus 10 sectors which cannot be used to store data when operating under DOS. In addition, one can only store 125 bytes in each of the remaining sectors. The last three bytes of each sector are used by DOS to keep track of what sector to go to next (i.e.; what sector has the next part of the file).

Note that when using DOS, you never have to even think about sector numbers. You simply give DOS a filename and it finds the appropriate sectors for you. This is very convenient, but it does require some disk space and if anything goes wrong with the disk directory (for example) you may be unable to access an entire file.

It is possible, however, to read or write any sector on the disk by simply using a few subroutines built into the Atari operating system. **Listing 1** is a short assembly language program, designed for use in a BASIC USR function, which shows how to do this. One simply sets a few parameters in the Device Control Block (DCB) and then jumps to the appropriate subroutine in ROM. **Listing 2** is a self-documented BASIC program illustrating the use of this USR function. A complete description of the device control block and its operation would be quite long. Anyone interested in further details should refer to Atari's Operating System Manual or "Outpost Atari" in *Creative Computing*, May 1982.

Warning!

Be *very careful* if you write sectors directly to a DOS disk. It is easy to make a mistake and change a sector in the middle of a file. DOS may then no longer be able to read the file. The one sector you don't have to worry about is sector 720 (since DOS cannot access it).

You may wonder why anyone would want to avoid using DOS and read or write specific disk sectors instead. I will list a few reasons below, and I am sure that you will think of many other possibilities.

1) Repair or recover damaged DOS files.

2) Write a sector-by-sector disk copying program. Note that the DOS disk copy command ("H") does not copy every sector. It only copies those sectors which the VTOC says are in use. If your VTOC is incorrect, DOS will not properly copy the disk. (Note that a simple sector copying program will not duplicate most commercially protected disks.)

3) Give your DOS disks a name. Because sector 720 is unused by DOS, you can use it in a program to give each disk a unique identifying code. You could store up to 128 characters in that sector when writing a data disk, and then check it later when reading the disk to see if the correct disk is in the drive.

4) Pack more data on a disk. As previously noted, DOS uses some disk sectors for its own bookkeeping purposes. By ignoring DOS, you can use these sectors for your purposes.

5) Move disk data directly to a desired location in memory. Remember that our USR function requires a parameter identifying an address in memory. Data is moved directly to or from that location. This can be useful in storing or retrieving high-resolution graphics data. For example, data can be moved directly from the disk to the screen memory location or vice-versa.

6) Use direct sector I/O to write a disk-based virtual memory system to extend apparent machine memory. This is a scheme used in various FORTH languages that have been implemented on the Atari.

7) If you are really ambitious, you can even write your own disk operating system! □

Listing 1.

```
0100 ; HOW TO LIVE WITHOUT DOS
0110 ; BY DAN HIGGINS
0120 ; ANALOG COMPUTING #17
0130 ;
0140 ; ASSEMBLY SOURCE CODE FOR USR FUNCTION
0150 ;
0160 DCB =   $0300    ; ADDR OF DEVICE CONTROL BLOCK
0170 DSKINV = $E453   ; OS ROUTINE TO R/W SECTORS
0180 ;
0190     *=  $0600    ; COULD GO ANYWHERE
0200 ;
0210     PLA          ; # OF ARGUMENTS
0220     PLA          ; HIGH BYTE OF BUFFER ADDR
0230     STA DCB+5
0240     PLA          ; LOW BYTE OF BUFFER ADDR
0250     STA DCB+4
0260     PLA          ; HIGH BYTE OF SECTOR #
0270     STA DCB+11
0280     PLA          ; LOW BYTE OF SECTOR #
0290     STA DCB+10
0300     PLA          ; HIGH BYTE OF R/W FLAG
0310     PLA          ; LOW BYTE OF R/W FLAG
0320     CMP #1       ; IS IT A 1?
0330     BNE READSEC  ; NO, SO READ A SECTOR
0340     LDA #$57     ; OS "WRITE SECTOR" COMMAND
0350     STA DCB+2
0360     BNE CONTINUE
0370 READSEC
0380     LDA #$52     ; OS "READ SECTOR" COMMAND
0390     STA DCB+2
0400 CONTINUE
0410     LDA #1       ; DRIVE #1
0420     STA DCB+1
0430     JSR DSKINV   ; PASS CONTROL TO OS
0440     RTS          ; THEN RETURN TO BASIC
0450 ;
0460     .END
```

•

Listing 2.

```
100 REM * HOW TO LIVE WITHOUT DOS
110 REM * BY DAN HIGGINS
120 REM * ANALOG COMPUTING #17
130 REM *
140 REM * THIS SAMPLE PROGRAM WILL
150 REM * SHOW YOU HOW TO READ OR
160 REM * WRITE DISK SECTORS WITHOUT
170 REM * GOING THROUGH DOS.
180 REM *
190 REM * A BASIC USR FUNCTION IS
200 REM * USED TO READ OR WRITE ONE
210 REM * DISK SECTOR AT A TIME
220 REM *
230 REM * FORM OF THE USR CALL IS:
240 REM *
250 REM * IO=USR(ADDR,BUFF,SECT,FLAG)
260 REM *
270 REM * SYNTAX:
280 REM *
290 REM * IO   = A DUMMY VARIABLE
300 REM * ADDR = ADDRESS OF THE
310 REM *        M/L SUBROUTINE
```

(Listing 2. continued next page.)

```
320 REM * BUFF = ADDRESS OF 128-BYTE
330 REM *        SECTOR BUFFER
340 REM * SECT = SECTOR NUMBER TO
350 REM *        ACCESS (1-720)
360 REM * FLAG = I/O DIRECTION FLAG
370 REM *        FLAG=0 TO READ
380 REM *        FLAG=1 TO WRITE
390 REM *
400 REM * FIRST WE MOVE THE MACHINE
410 REM * LANGUAGE SUBROUTINE INTO A
420 REM * STRING (SECRW$)
430 REM *
440 DIM SECRW$(44)
450 FOR I=1 TO 44
460 READ X
470 SECRW$(I)=CHR$(X)
480 NEXT I
490 REM *
500 REM * NOW WE RESERVE A 128-BYTE
510 REM * BUFFER AREA TO HOLD THE
520 REM * SECTOR DATA
530 REM *
540 DIM BUF$(128)
550 REM *
560 REM * THE FOLLOWING LINES WILL
570 REM * WRITE THE CONTENTS OF BUF$
580 REM * TO DISK SECTOR 720
590 REM *
600 BUF$="TESTING 1 2 3 4"
610 IO=USR(ADR(SECRW$),ADR(BUF$),720,1
)
620 BUF$="                "
630 PRINT BUF$:REM * BUF$ CLEARED
640 REM *
650 REM * NOW WE'LL READ SECTOR 720
660 REM * BACK INTO BUF$
670 REM *
680 IO=USR(ADR(SECRW$),ADR(BUF$),720,0
)
690 PRINT BUF$:REM * BUF$ RESTORED!
700 END
710 REM *
720 REM * DATA FOR MACHINE LANGUAGE
730 REM * SUBROUTINE (SEE LISTING 1)
740 REM *
750 DATA 104,104,141,5,3,104,141,4,3,1
04
760 DATA 141,11,3,104,141,10,3,104,104
,201
770 DATA 1,208,7,169,87,141,2,3,208,5
780 DATA 169,82,141,2,3,169,1,141,1,3
790 DATA 32,83,228,96
```

•

CHECKSUM DATA
(see p. 15)

```
100 DATA 558,175,993,280,1,455,170,66,
295,506,568,358,279,715,285,5704
250 DATA 111,291,364,297,228,243,984,6
29,656,551,567,531,790,782,302,7326
400 DATA 744,31,557,286,336,150,700,88
4,758,304,468,404,907,288,50,6867
550 DATA 294,988,854,919,306,91,88,247
,347,293,343,217,302,106,541,5936
700 DATA 38,286,735,927,295,493,526,32
2,274,591,4487
```

•

ATARI®

Printers/Etc.

600XL .... **CALL**
1200XL .... **CALL***
*Reflects $100 Atari Rebate

1050 DRIVE $335
1025 PRINTER $399
1020 COLOR PTR $219
1027 PRINTER $279
1010 RECORDER $72
MICRO PERIF. MODEM/INFC $129
MICROBITS INFC $78
80 COLUMN BD $249
TECHNICAL NOTES.... $25
REAL TIME CLK $38
AXIOM PRINTER INFC $80

DRIVES

ASTRA 1620 .... $469
RANA 1000 .... $295
PERCOM 88-S1 ..... $298
PERCOM 40-S1 ..... $418
PERCOM 40-S2 ..... $718
PERCOM 44-S1 ..... $510
PERCOM 44-S2 ..... $929

MEMORIES

48K RAM (INTEC) $95
64K RAM (INTEC) .. $119
48K RAM (MOSAIC) .. $109
64K RAM (MOSAIC) .. $145
128K RAM DISK $299
32K RAM (MOSAIC) ... $68

GEMINI 10X .. $279
GORILLA $199
PROWRITER . $345
SMITH TPI ... $488

CITOH
Prowriter $345
Prowriter II $629
Starwriter $1149
Printmaster $1448

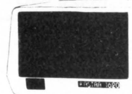
NEC
8023 A-C $409
3510 .... $1375
3530 .... $1579
3550 .... $1779
7710/7730 .... $1998

SILVER REED P $669
QUME 11/40 + ... $1299

OKI-DATA
Microline 82A ......
Microline 83A ......
Microline 84P ......
Microline 92 ......
Microline 93 ......
CALL

DIABLO
620R $939
630R $1719

✶ASTRA 1620



DISK DRIVE SYSTEM

MORE FOR YOUR MONEY
DOUBLE OR SINGLE DENSITY
TWO DRIVES

SPECIAL $469

MONITORS

AMDEK
Color I $289
V300 .... $139
V300A $149
Color II $449

NEC
GRN (JB1260) $115
GRN (JB1201) $155
Color Composite ... $298
RGB Color $598

MODEMS

HAYES
Smartmodem $209
Smartmodem 1200 .. $498
Micromodem II $259

NOVATION
J-Cat $99
Apple Cat II $259
D-Cat $149

ANCHOR AUTOMATION
Mark I or II Modem ... $78

Q-BERT $33
POPEYE $33

D = DISK
R = ROM (Cartridge)
C = CASSETTE

COSMIC™

ATARI SOFTWARE

ADVENTURE INT'L
Adv. 1-12 each (C) $18
Preppie (C/D) $20
Preppie II (C/D) $23
Diskey (D) $33
Sea Dragon (C/D) $23
Bug Off! (C/D) $20
Tutti-Frutti (C/D) $17
Saga 1-3 (ea.) (D) $27
Lunar Lndr . (C) $11 (D) $15
Galactic Empire (C) .. $14
Galactic Trader (C) ... $14
Stratos (C/D) $23
Rear Guard (C) $14 (D) $17

APX
Eastern Front (C/D) .. $23
747 Land Sim. (C/D) .. $17
Fig-Forth (C) $30
Family Cash Flow (D) . $17
Downhill (C/D) $17
Avalanche (C/D) $17
Outlaw/Howitzer (C/D) $17
Salmon Run (C/D) $17
Data Management (D) $17
Galahad/Holy Grail (D) $21
Adv. Music System (D) $21

ATARI INC.
Microsoft Basic II (R) . $62
Mickey in Great Outdoors (C/D) $36
Paint (D) $30
Speed Reading (C) ... $54
Qix (R) $30
Dig Dug (R) $30
Atari Writer (R) $68
Time Wise (D) $23
Visicalc (D) $139
Juggles House (C/D) . $22
Juggles Rnbw (C/D) . $22
Pilot (Home) (R) $55
Galaxian (R) $30
Defender (R) $30
ET $34
Microsoft Basic (D) .. $62
Macro Ass. & Edit (D) . $62
Assembler Editor (R) . $42
Basic Cartridge (R) .. $45
Pac Man (R) $30
Centipede (R) $30
Caverns of Mars (D) .. $28
Star Raiders (R) $30
Conv. Lang. Ea. (C) ... $42
Music Composer (R) .. $31
Super Breakout (R) ... $26
My First Alphabet (D) . $26
Prog.2 & E (ea.)(C) $21
Word Processor (D) . $102
Pilot (Educ.) (R) $92
Touch Typing (C) $19
Home File Mngr (D) .. $36
Bookkeeper (D) $102
Family Finance (D) ... $35
Prog. 1 (C) $18
Scram (C) $18
Asteroids (R) $27
Space Invaders (R) ... $27
Missile Command (R) . $27
Telelink (R) $21
Superman III (R) $34
Basketball (R) $24
Donkey Kong (R) $34

AVALON HILL
VC (D) $17
B-1 Nuc. Bomber (C) .. $12
Legionnaire (C) $23
Empire of Overmind .... (D) $23 (C) $20
Tanktics ... (D) $20 (C) $17
Comptr Stock & Bonds .... (D) $17 (C) $14
Dnieper River Line (D) $20
Voyager (D) $17
Controller (D) $20
GFS Sorceress .... (D) $23 (C) $20
Telegard .. (D) $19 (C) $16
Vorrak (D) $18
Lord of Karma (C) $14

AVALON HILL(Cont'd)
Moon Patrol (C) $17
Planet Miners (C) $12
Conflict 2500 (C) $12
Space Station Zulu .... (D) $17 (C) $14
Flying Ace . (D) $21 (C) $18
Gypsy ..... (D) $18 (C) $15
Galaxy .... (D) $17 (C) $14
Guns of Fort Def. (C) . $14
Nuke War (C) $12
Andromeda Conquest .... (D) $16 (C) $13
Close Asslt (D) $23 (C) $20

BRODERBUND
Sky Blazer (D) $22
Bank St. Writer (D) ... $46
A.E. (D) $23
Arcade Machine (D) .. $39
Choplifter .. (D) $23 (R) $29
Labyrinth (C/D) $20
Serpentine .. (D) $23 (R) $27
Steller Shuttle (C/D) .. $17
Apple Panic (C/D) $20
Genetic Drift (C/D) ... $20
David's Midnight (D) .. $23
Sea Fox (D) $20
Track Attack (D) $20
Operatn Whrlwnd (D) . $23
Match Boxes (C/D) ... $20

CDY CONSULTING
Pogoman (C/D) $27
Mad-Netter (C/D) $23
It-Is-Balloon (C/D) $23

CBS
Mountain King (R) ... $27
Boulders & Bombs (R) $27
Krazy (each) (R) $27

CONTINENTAL SOFT.
Home Accountant (D) $48
Tax Advantage (D) ... $39

DATA MOST
Pig Pen (D) $20
Mating Zone (D) $20
Roundabout (D) $20
Bilestoad (D) $20
Night Raiders (D) $20
Monster Smash (D) ... $20

DATASOFT
Text Wizard (D) $65
Graphic Master (D) ... $27
Micro Painter (D) $23
Lisp Interpreter (D) .. $79
Graphics Gen.(D) $17
Basic Compiler (D) ... $65
Zaxxon (C/D) $27
Teletalk (D) $34
Pac. Coast Hwy (C/D) . $20
Clowns/Balloons (C/D) $20
Spell Wizard (D) $53
Canyon Climber (C/D) $20
Sands of Egypt (D) ... $27
O'Riley's Mine (C/D) .. $23
Rosen's Brigade (C/D) $23
Moon Shuttle (D) $27

DON'T ASK
Sam (D) $39
P.M. Animator (D) $23
Teletari (D) $27
Poker Sam (D) $17
Word Race . (D) $17 (C) $14

EDU-WARE
Prisoner II (D) $27
Spelling Bee (D) $27
Rendevous (D) $27
Compu/Read .... (D) $21 (C) $15
Compu/Math-FR .... (D) $27 (C) $21
Compu/Math-Dec .... (D) $27 (C) $21

EDUCATIONAL SOFT.
Tricky Tutorial 1,2,3 or 4 (C/D) $15
Tricky Tutorial 5,6 or 7 (C/D) $22

SPECIALS

Gemini 15X Printer $399
Axiom AT-100 Printer (with interface $229
Astra Double Density Dual Drive $469
Rana 1000 Drive $319
Bit-3 80 Column Board $245
Mannesmann Talley 160L Printer $589
Atari 400 Keyboard (In Home) $35
Programmer Kit .. $48 Entertainer Kit ... $64
Wico Joystick ... $23 Wico Trackball ... $49

COSMIC COMPUTERS
UNLIMITED

727 BREA CANYON RD., SUITE 16
WALNUT, CA 91789

ORDER LINES OPEN MON-SAT 8 am - 8 pm

(800) 626-7642

PLEASE FOR ORDERS ONLY
SORRY, NO COD'S

(714) 594-5204

FOR TECHNICAL INFO, ORDER INQUIRIES, OR FOR CALIFORNIA ORDERS

Add $2.50 shipping per software order in continental U.S. Add $5.00 shipping per software order for AK, HI, FPO-APO. Add $10.00 or 15% (whichever is greater) per software order for non-U.S. Call for cost of hardware shipping. Calif. residents add 6½% sales tax. Cashiers checks or money orders filled within 24 hours for items in stock. Personal checks require 4 weeks to clear. MasterCard and Visa OK for software only within continental U.S., add 3% surcharge. Include card no., expiration date and signature. Due to our low prices, all sales are final. All defective returns must have a return authorization number. Please call to obtain one before returning goods for replacement or repair. Prices & availability subject to change.

EPYX-AUTO. SIMULATION
Rescue at Rigel (C/D) . $20
Datestones/Ryn (C/D) $14
Upr Rchs Apshai (C/D) $14
Ricochet (C/D) $14
Keys/Acheron (C/D) .. $14
Curse of Ra (C/D) $14
Danger/Drindisti (C/D) $14
Jumpman (C/D) $27
Escp Vulcans Isle (D) . $20
Crypt of Undead (D) .. $20
Nightmare (D) $20
Armor Assault (D) $27
Monster Maze (R) $27
Alien Garden (R) $27
Plattermarna (R) $27
Morloc's Tower (C) ... $14
King Arthur's Heir .... (D) $20 (C) $14
Hellfire Warrior (C/D) . $27
Invasion Orion (C/D) .. $17
Temple of Aps. (C/D) . $27
Star Warrior (C/D) $27
Dragon's Eye (D) $20
Crush Crumble (C/D) . $20

GEBELLI
Firebird (R) $27
Embargo (R) $30
Candy Factory (D) $20
Match Racers (C/D) ... $20
Dr. Goodcode (D) $20

HAYDEN
Go .... (D) $23 (C) $20
Sargon II .. (D) $23 (C) $20
Bulldog Pinball (C) ... $20

HES
Coco (C/D) $34
Coco II (C/D) $27

INFOCOM
Suspended (D) $34
Zork I, II or III (D) $27
Starcross (D) $27
Deadline (D) $34
Witness (D) $34

INHOME
Baseball .. (R) $23 (D) $23
Crypts of Terror .... (D) $23 (C) $20

INNOVATIVE DESIGN
Pool 1.5 (D) $23
Juggler (D) $20
Speedway Blast .... (D) $20 (R) $27
Pool 400 (R) $27

JV SOFTWARE
Jrny to Plnts (C/D) $20
Action Quest (C/D) ... $20
Ghost Encount. (C/D) . $20

LJK
Letter Perfect (D) ... $104
Data Perfect (D) $74
Letter Perfect (R) $137
Edit 6502 (R) $137

MONARCH DATA SYS.
ABC Compiler (D) $48

ODESTA
Chess (D) $45
Checkers (D) $34
Odin (D) $34

OPTIMIZED SYSTEMS
C-65 (D) $58
Bug-65 (D) $23
Max-65 (D) $58
Basic A + (D) $58
Action (D) $65
Speed Read Plus (D) .. $40

PHOENIX
Birth of Phoenix (D) .. $16
Adv. In Time (D) $20

QUALITY
Fastgammon (C) $14
Name That Song .... (D) $13 (C) $11
Starbase Hyperion .... (D) $17 (C) $14
Ali Baba/40 Thvs (D) .. $20
Jeeper Creepers (D) .. $20

ROKLAN
Gorf ...... (D) $27 (R) $30
Wizard/Wor . (D)$27 (R)$30
Dlx Invaders (D)$23 (R)$27
Anti Sub Ptrl (D)$20 (C)$14
Telecom (D) $49
Space Journey (R) ... $30

SENTIENT
Gold Rush (D) $23
Cyborg (D) $23

SIERRA ON-LINE
Mission Asteroids (D) $17
Ulys.& Gldn Fleece (D) $27
Crossfire .. (D) $20 (R) $23
Mouseattack (D) $23
Jawbreaker (D) $20 (R) $23
Threshold (D) $27
Softporn (D) $20
Ultima II (D) $39

SIERRA ON-LINE
Marauder (D) $23
Lunar Leeper (D) $20
Wiz & Princess (D) ... $22
Frogger (C/D) $23

SIRIUS
Alpha Shield (R) $27
Wavy Navy (D) $23
Bandits (D) $23
Space Eggs (D) $20
Sneakers (D) $20
Way Out (D) $27
Type Attack (D) $27
Repton (D) $27
Critical Mass (D) $27
Fast Eddy (R) $23
Worm War (R) $23

SPINNAKER
Snooper Troop 1,2 (D) . $30
Kindercomp (D) $20
Rhymes & Riddles (D) $20
Hey Diddle Diddle (D) . $20
Srch Amzng Thngs (D) $27
Story Machine (D) $23
Face Maker (D) $23

STRATEGIC SIM.
Cosmic Balance (D) .. $27
Cosmic Balance II (D) $27
Tigers In Snow (C/D) . $27
Battle of Shiloh (C/D) . $27
Battle of Norm. (C/D) . $27
Cytron Masters (D) ... $27
Knights/Desert (C/D) . $27
Combat Leader (C/D) . $27

SUBLOGIC
Pinball (C/D) $20

SWIFTY
J. White Music Lessons .... (C/D) $20
Space Shuttle (D) $20

SYNAPSE SOFTWARE
File Mngr 800 + ... $65
Protector II (D) $23 (R) $29
Shamus ... (D) $23 (R) $29
Fort Apocalypse (C/D) $23
Shamus II (C/D) $23
Necromancer (C/D) ... $23
Pharoh's Curse (C/D) . $23
Slime (C/D) $23
Page 6 (D) $23
Reptilian (C/D) $23
Picnic Paranoia (C/D) . $23
Claim Jumper (C/D) .. $23
Drelbs (C/D) $23
Shadow World (C/D) .. $23
Survivor (C/D) $23
T.N.T. (C/D) $23
N.Y.C. (C/D) $23
Slamball (C/D) $23
Blue Max (C/D) $23
Nautilus (C/D) $23
Chicken (R/D) $23
Quasimodo (C/D) $23
River Quest (C/D) $23

SYNERGISTIC
Programr Wkshp (D) .. $23
Graph Workshop (D) .. $27
Disk Workshop (D) ... $23
Worlock's Revenge (D) $23

THORN EMI
Soccer (R) $34
Jumbo Jet (R) $34
Submarine Comm. (R) $34
Kick Back (R) $34
Pool (C) $22
Humpty Dumpty (C) .. $22
Hckry Dckry Dck (C) .. $22
Darts (C) $22

USA
Atari World (D) $39
3-D Sprgrphcs (C/D) .. $27
Survival Adv. (C/D) ... $17

VERSA
Mind Bgglrs I (D)$14 (C)$12
Globe Master (D) $20

MISCELLANEOUS
Financial Wizard (D) .. $41
Castle Wolfenstein (D) $20
Master Type (D) $27
Astro Chase (D) $22
Miner 2049er (R) $34
Cypher Bowel (C) $33
Bug Attack (C/D) $20
Raster Blaster (D) $20
Air Strike (C/D) $27
Prism (D) $18
Megalegs (C) $23
Cap'n Cosmo (D) $19
Spy's Demise (D) $14
Galac. Chase (D)$20 (C)$18
Kid Grid (C/D) $20
Battle Trek (D) $20
Snapper (D) $19
T&F-PMP Property Management (D) .. $149
Millionaire (D) $39

INTRODUCTION TO ACTION!

Part 1.

by Clinton Parker

This is the first of a 2-part series that will introduce you to the Action! programming language, using a short example program that draws kaleidoscopic patterns on the screen. There's an old saying about fooling people which, unfortunately, holds true for trying to please people as well. The problem in my case is that different readers have different levels of experience. I hope this series will please all of you at least some of the time.

Action! is a true compiled language, whereas Atari BASIC is an interactive interpreter. In both cases, the ultimate goal is to translate programs from a human-readable form into something that the computer can understand. The difference is that Action! only performs this translation once, whereas BASIC does it repeatedly. The process is similar to having a speech translated from German to English once and then reading it many times in English (Action!), as opposed to having someone translate the speech to English every time it is read (BASIC). Because Action! statements don't have to be translated each time, they execute much faster.

Action! has three types of numeric variables (BYTEs, CARDinals and INTegers), which are easier for the computer to deal with than the floating-point numbers always used by Atari BASIC. This also contributes to faster program execution, but costs you in terms of flexibility (no fractions or very large numbers) and simplicity (you must declare variables so that the compiler will know what type they are).

BYTE variables can represent numbers from 0 to 255. CARDs can represent numbers from 0 to 65535, and INTs can represent numbers from –32768 to 32767. Referring to **Listing 1**, the lines:

```
CARD period, npts
BYTE x0, y0, x1, y1, ATRACT=77
BYTE CH=764
```

are called variable declarations. Note that the BYTE variable **ATRACT** is defined to reference location 77 in memory, and that variable **CH** references location 764. More on these later.

In addition to the three basic types described above, Action! allows ARRAYs, POINTERs and user-defined TYPEs (records). The following line:

```
TYPE REC=[CARD cnt,ax,bx,cx,ay,by,cy]
```

is a TYPE declaration named **REC**, and:

```
REC p, e
```

is a declaration of two variables (**p** and **e**) of type **REC**. Each of these variables contain all of the variable fields specified in the declaration of **REC**. Fields of record variables are referenced by first giving the record variable name, then a '.' (period), followed by the field name.

The lines:

```
p.ax = 5221   p.bx = 64449   p.cx = 3
p.ay = 57669  p.by = 64489   p.cy = 3
```

are examples of assignment statements using record fields.

Action!'s assignment statements are very, very similar to BASIC assignments. The IF structure is also similar to BASIC's, with two important exceptions. First, BASIC conditional statements must fit in the same logical line as the IF. Action! lets you include as many statements following the THEN as you like, because the compiler treats End-Of-Line characters the same as spaces or colons. The Action! keyword FI (IF spelled backwards) is used to end a list of statements following the corresponding THEN.

Second, Action! makes it possible to execute a list of statements if the condition following an IF is *false*. This is done by placing the keyword ELSE where the FI would normally go, followed by the list of statements for the ELSE, and finally an FI to terminate the structure. ELSE is not used in **Listing 1**, so don't be concerned if you don't see one.

Action! loops are used to execute a group of statements repeatedly. A simple loop is specified by the keyword DO, followed by a list of statements and ending with the keyword OD (DO spelled backwards). The effect is similar to a group of BASIC statements with a GOTO <first statement> as the last statement in the group. You can provide control information to specify how many times an Action! loop is to be repeated. One loop control structure — FOR/TO — is very similar to the FOR structure in Atari BASIC. The differences are that, in Action!, the end condition is always tested *before* the statements within the loop are executed, which means that the loop may never be executed. BASIC always executes a FOR/NEXT loop at least once. Additionally, the STEP increment may only be positive in Action!, whereas BASIC allows both positive and negative STEPs. The other two Action! control structures, WHILE and UNTIL, will be discussed later.

PROCedures.

An Action! PROCedure is roughly the same as an Atari BASIC subroutine. One distinction is that it's possible to pass arguments to an Action! PROCedure. If you've ever called a function in BASIC, then you have already used argument passing without even realizing it. In the BASIC line:

```
A=SIN(X)
```

X is the argument to the function call SIN().

The **Listing 1** lines:

```
MoveBlock(e, p, REC)
  Gen(p)
```

are examples of PROC calls. Note that the Action! compiler makes no distinction between user-defined PROCs and system subroutines. Thus, the PROC calls:

```
Graphics(24)
SetColor(1,0,14) : SetColor(2,0,0)
```

are similar to the BASIC statements:

```
GRAPHICS 24
SETCOLOR 1,0,14:SETCOLOR 2,0,0
```

This gives us a nice, uniform PROCedure-calling mechanism, and provides an easy method for users to provide their own versions of system routines.

PROCedure declarations tell the Action! compiler the name by which the PROC can be called, the arguments and variables which are unique to that PROC, and which statements are to be executed when the PROC is called. In our **Listing 1** example, everything between:

```
PROC Gen(REC POINTER r)
```

and

```
PROC Kal()
```

constitutes the declaration for the PROCedure **Gen()**.

Gen() has one argument, **r**, which is a POINTER variable of type **REC** (a user-defined TYPE).

The line:

```
BYTE x0, y0, x1, y1, ATRACT=77
```

declares a number of *local* variables that are only used in **Gen()**. They can not be accessed by any other PROCedure in the program (**Kal()** in this case). However, the *global* variable **period** (which was declared at the beginning of the program) can be used by either PROCedure.

The RETURN statement at the end of the declaration for **Gen()** is the same as a RETURN statement in BASIC, and causes execution to jump back to the point from which the PROCedure was called. The last procedure declared in a program is the one which will be called first when the program is started (**Kal()** in this example). If you don't quite follow all of this, don't worry; things should get clearer as we walk through the example.

Walking through.

As stated earlier, **Listing 1** draws kaleidoscopic patterns on the screen. This is done by repeatedly calling the PROCedure **Gen()**. The **Gen()** statements:

```
r.ax = (r.ax + r.bx) ! r.bx
r.ay = (r.ay + r.by) ! r.by
```

generate new values for **ax** and **ay** (fields of record **r**, passed to the **Gen()** PROCedure). These values are used to calculate **x0** and **y0** as follows:

```
x0 = r.ax RSH 9
y0 = r.ay RSH 9
```

Without going into details about bit arithmetic and operations, the RSH 9 statements have the effect of dividing **r.ax** and **r.ay** by 512 (but do it much faster than a "real" divide). The reason for dividing by 512 is to get values in the range 0-127, so that they can be plotted in graphics mode 24.

The IF statement:

```
IF x0 <= y0 AND y0 < 96 THEN
  .
  .
FI
```

determines if any points are to be plotted. The check for **y0 < 96** assures that the points won't overlap when we calculate **x1** and **y1**:

```
x1 = 191 - x0
y1 = 191 - y0
```

The value of 191 was chosen since it is the maximum y-value you can plot in graphics mode 24.

The Plot calls following these two statements display all eight combinations of **x0**, **y0**, **x1**, and **y1**. The +64 in each call centers the display on the screen, since there are 128 more points in the X direction than there are in the Y direction.

If you're curious about how this plotting algorithm works, choose your own values for **x0** and **y0** (21 and 55, for example). Calculate **x1** and **y1** from the formula above (170,136). Finally, calculate all of the points that will be plotted (don't add in the 64; it makes things easier to see). Our example would yield coordinates of (21,55), (21,136), (55,21), (55,170), (170,55), (170,136), (136,21) and (136,170). If you plot these on a piece of graph paper with 0,0 in the upper left corner and 191,191 in the lower right, you'll see that they are symmetric about the center.

The only part of **Gen()** not explained yet is:

```
r.cnt == -1
IF r.cnt = 0 THEN
  .
  .
FI
```

The first statement decrements the **cnt** field of **r**, and the IF statement body is executed when **cnt** reaches zero.

The statements:

```
r.bx = (r.bx + r.cx) ! r.cx
r.by = (r.by + r.cy) ! r.cy
```

calculate new values for **bx** and **by**, which cause the **ax** and **ay** calculations to change in the future as well.

The line:

```
r.cnt = period
```

resets **cnt** so that it can count down to zero again. Finally,

```
ATRACT = 0
```

keeps the screen from going into attract mode. Note that **ATRACT** was declared to be at location 77. This is the memory location used by the OS to determine if attract mode is on or off.

A look at Kal().

Now you understand (I hope) how the **Gen()** procedure works. So let's look at **Kal()** and see how it uses **Gen()**.

The first three **Kal()** statements:

```
Graphics(24)
SetColor(1,0,14) : SetColor(2,0,0)
```

set up graphics mode 24, with white dots on a black background. The next group:

```
persistence = 2500
period = 10000  p.cnt = period
p.ax = 5221     p.bx = 64449  p.cx = 3
p.ay = 57669    p.bx = 64489  p.cy = 3
```

sets the initial values that control the pattern generation of **Gen()**. You can change these to generate your own patterns. As stated above, **ax**, **ay**, **bx**, **by**, **cx** and **cy** are used to calculate the points to be plotted. The value for **period** determines how frequently the pattern will change. The value for **persistence** determines how much of the pattern will be on the screen at once.

You may be saying at this point, "Hold on there! If you don't erase any points, the screen will just turn white," and you would be right. That's the reason for:

```
MoveBlock(e, p, REC)
```

and why **Gen()** is passed a record argument. It turns out that, depending on the value of **color**, **Gen()** will either plot or erase points on the screen. The **p** record will be used for plotting, and the **e** record will be used for erasing. MoveBlock makes a copy of **p** (all the fields) in **e**, because when a record variable is referenced without a field, the address of the record is used. When a type name is referenced, the size in bytes of the type is used. Thus, MoveBlock is being called with the address of records **e** and **p**, and the size of the record. Initially both **p** and **e** will have the same values. Here is how **p** and **e** are used:

```
WHILE CH = 255 DO
  color = 1  Gen(p)
  color = 0  Gen(e)
OD
```

First, **color** is set to one (plot points) and **Gen()** is called with **p** as an argument (remember, this passes the address of **p**, a POINTER, to the **Gen()** procedure). Next, **color** is set to zero (erase points) and **Gen()** is called with **e** as an argument. Since both **p** and **e** start out the same, what happens is that **Gen(p)** draws some points on the screen and **Gen(e)** erases them. That keeps the screen from turning white.

The sequence will keep repeating as long as **CH** equals 255. **CH** was declared to be at address 764, the location that the OS stores the internal value for the last key pressed. It is set to 255 by the keyboard handler after a key is processed. Thus, as long as no key is depressed, **CH** will equal 255. As soon as a key is depressed, it will contain the code for the last key (will no longer equal 255) and the loop will terminate, causing:

JOUST
ATARI, Inc.
P.O. Box 61657
Sunnyvale, California 94086
16K Cartridge $39.95

by Joel Gluck

I'll admit it. **Joust**, by Williams, is probably my favorite arcade game. No other video game provides such playful freedom, marvelous two-player interaction, and hilarious physical antics as **Joust**.

A description of **Joust** can be deceptively simple. Briefly put, you control a bird upon which your knight rides, flying about by means of *flapping* with the control button and moving the control stick right and left. "Jousting" with one of the many opponent flying knights on the screen entails colliding with them; whoever's mount is highest at the moment of collision is the victor. If you succeed, the enemy bird flys off riderless, leaving an egg behind, which you must retrieve before it hatches into a new opponent. If the enemy knight wins, you lose your life (but do not despair; you have several lives).

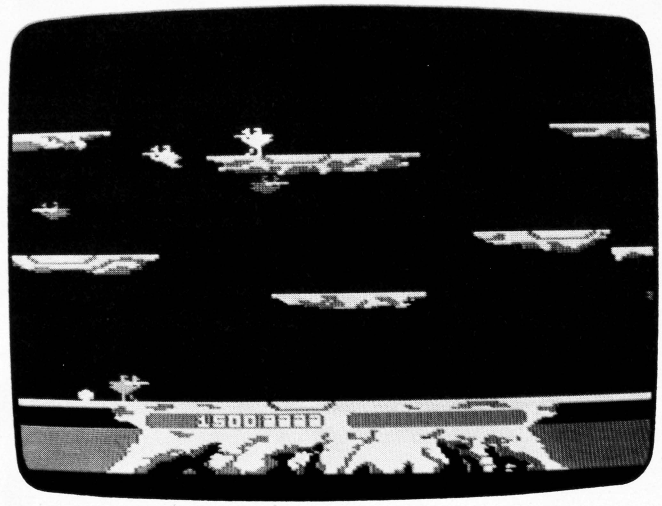
Of course, there are complexities in the game. For one, there are three kinds of enemy knights — and, if an egg hatches, it hatches into a knight of a more dangerous kind. There are other hazards: the Lava Troll awaits in the pool of fire at the bottom on the screen, ready to pull you in if you fly too close; and deadly Pterodactyls frequently prowl the screen, usually appearing at the end of a round.

I love the arcade version of **Joust**, and worried that the new **Joust** cartridge for the Atari Home Computer would be inferior. My opinion, now that I've played the game, is that the programmers of Atari **Joust** did an admirable job, considering the memory limitations of a home computer.

For starters, the screen is beautiful. I was shocked to see that the original **Joust** playfield had been almost exactly duplicated, in every color and detail. The game is also faithful to the arcade version in most other details of play, with a few minor exceptions. For example, when an egg hatches, in the arcade version a knight appears and waits for its mount to fly onto the screen and pick it up. In the Atari version, both knight and mount hatch as one.

The best reason for me to recommend Atari **Joust** is that it is fun to play, especially as a two-player game. However, for the record, I do have a few complaints:

1. Control over your knight (with the joystick) is not as precise as in the arcade version. The probable reason for this is that the game had to be designed so that one would not have to push the trigger as often as one has to hit the flap button in the arcade version. Also, I find it difficult to stop or change directions quickly, something I find easy in arcade **Joust**.
2. Atari **Joust** proceeds from one round to the next too quickly. There should be time to position your knight properly, as in the arcade version.
3. At the end of the game, at the exact moment when you ordinarily glance down to see your final score, Atari **Joust** immediately switches to the title screen. This is very annoying.



Joust.

These complaints are relatively minor, however. The game is fun, and that's what counts. I've seen some bad arcade adaptations in my time, and Atari's new **Joust** isn't one of them. □

```
CH = 255 : Graphics(0)
  RETURN
```

to be executed. This sets **CH** back to 255 so that the keyboard handler won't think a key has been depressed, and restores graphics mode 0 before returning to the Action! monitor.

I'll bet you're wondering why I didn't mention:

```
color = 1
FOR npnts = 1 TO persistence DO
  Gen(p)
  UNTIL CH#255
OD
```

yet. It's there for a reason. If you execute the loop below it, only one set of points will be displayed at a time. Although this is somewhat interesting, it isn't what I intended. The FOR loop causes **"persistence"** sets of points to be generated without any being erased (note that only **Gen(p)** is called, with **color** equal to one). So when the WHILE loop below this is reached, the call to **Gen(e)** will erase points that were plotted **"persistence"** interactions earlier. The values of **p** will always be **"persistence"** interactions ahead of **e**. Thus, you'll always have at most **"persistence"** sets of points on the screen at any given time.

The UNTIL at the end of the loop serves the same purpose as the WHILE described earlier. The only difference is that an UNTIL loop repeats as long as the condition is false (the inverse of WHILE). That's why **CH** is tested to not equal 255 (inverse of equal in WHILE).

Those of you who have an Action! cartridge should try this program. It's very small and easy to enter. The first thing you'll notice is that it doesn't run especially fast. This is mainly due to the fact that it is using the Atari operating system's PLOT subroutine. In Part II of this series, I'll discuss some things you can do to speed it up. You may also wish to adjust the colors on your TV set or monitor to get the best-looking patterns. □

Action! listing.

```
;         KAL.ACT

; ANALOG Computing #17
; Copyright 1984 BY Clinton Parker
; All Rights Reserved

; last modified January 11, 1984

; Global variables
```

```
TYPE REC=[CARD cnt,ax,bx,cx,ay,by,cy]
REC p, e
CARD period, npts, persistence

PROC Gen(REC POINTER r)

  BYTE x0, y0, x1, y1, ATRACT=77

; get new a

  r.ax = (r.ax + r.bx) ! r.bx
  r.ay = (r.ay + r.by) ! r.by

  r.cnt == -1
  IF r.cnt = 0 THEN ; get new b
    r.bx = (r.bx + r.cx) ! r.cx
    r.by = (r.by + r.cy) ! r.cy
    r.cnt = period
    ATRACT = 0 ; turn off attact mode
  FI

  x0 = r.ax RSH 9
  y0 = r.ay RSH 9
  IF x0 <= y0 AND y0 < 96 THEN
    x1 = 191 - x0
    y1 = 191 - y0
    Plot(x0+64, y0) : Plot(x0+64, y1)
    Plot(y0+64, x0) : Plot(y0+64, x1)
    Plot(x1+64, y0) : Plot(x1+64, y1)
    Plot(y1+64, x0) : Plot(y1+64, x1)
  FI

RETURN


PROC Kal()

  CHAR CH=764

  Graphics(24)
  SetColor(1,0,14) : SetColor(2,0,0)

; change for different patterns:

  persistence = 2500
  period = 10000  p.cnt = period
  p.ax= 5221   p.bx=64449   p.cx=3
  p.ay=57669   p.by=64489   p.cy=3

; copy plot record to erase record

  MoveBlock(e, p, REC)

; handle persistence

  color = 1
  FOR npts = 1 TO persistence DO
    Gen(p)
    UNTIL CH#255
  OD

; draw patterns until key drepressed

  WHILE CH = 255 DO
    color = 1  Gen(p)
    color = 0  Gen(e)
  OD

; ignore key and restore screen

  CH = 255 : Graphics(0)

RETURN
```

•

SILICON WARRIOR
EPYX, Inc.
1043 Kiel Court
Sunnyvale, California 94089
16K Cartridge $39.95

by James Trunzo

Silicon Warrior, the newest cartridge release by Epyx for Atari computer systems, takes a simple tic-tac-toe playing scenario and turns it into a challenging strategy/action game that will please both the intellect and the arcader alike. Set in the Silicon Valley in the year 2084, **Warrior** depicts the conflict between (now get this!) the House of Peanut, the House of Apple, the House of Adam and the House of Pong! Subtle, huh? Each house is represented by a warrior whose job it is to "program" five chips in a row (horizontally, vertically or diagonally), and defend them for a short time period to achieve a victory. Now while this may sound like a simple task, it is in concept only.

Play begins with the selection of the game's parameters. **Warrior** allows up to four combatants per contest in any combination of human and/or computer players (the Atari 400 and 800, with their four joystick ports, will allow four human players while the new XL systems would be limited to two). Players may select from seven levels of difficulty. Furthermore, each level may be played at one of three speeds — slow, medium or fast. Obviously, there are a tremendous number of playing combinations that may be designed which gives **Silicon Warrior** almost unlimited flexibility and variation, insuring that the game will not become stale after repeated plays.

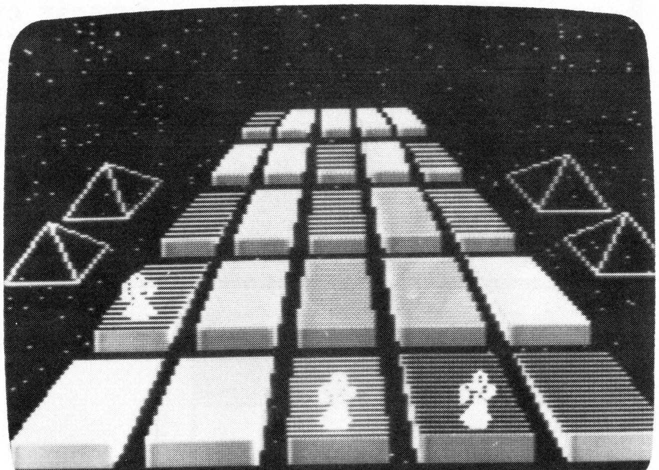
Once parameters are selected, the game begins. The playfield is simple in the extreme. It's made up of twenty-five rectangular "chips" and four power pyramids. The chips are all "unprogrammed" (blank) at the beginning and, as mentioned, the player's task is to program five chips in a row. To do this, you must land on a chip, which will immediately change to the player's color. What makes this difficult to accomplish, however, are any or all (depending on the difficulty level selected) of the following obstacles:

Enemy warriors: Warriors from opposing houses are competing for chips and moving at the same time you are. The computer plays a strong game on all levels, and will block you whenever it can. The enemy can also land on a chip which you have programmed and change it back into a blank chip, thwarting your carefully planned strategy. In addition, warriors can fire lasers (so can you) which drain your power and eventually force you back to your power pyramid for a recharge. Of course, all the while that you're inside your pyramid, the other warriors are programming new chips and de-programming yours.

Shields: On higher levels of play, all warriors become armed with shields that protect them from enemy laser blasts. This option adds another layer of strategy to play by introducing the element of defense to the game.

Black Hole Glitches: Occasionally, a glitch will occur, turning one of the chips back. Any warrior occupying the chip or teleporting to it when it turns black is instantly returned to his power pyramid.

You can easily see that, with all of the factors mentioned above coming into play along with the speed variations and multiple opponents, **Silicon Warrior** can certainly represent a significant challenge to both mind and hand.



Although **Silicon Warrior** consists of only one screen, don't assume that the game lacks in the area of graphics, sounds or special effects. Consider that each warrior is a specific color and that each chip he programs adopts that color; consider also that a distinct tone emanates from a chip as it is programmed. When you blend these two elements together, along with the fact that the actions of the four warriors take place simultaneously and continuously, you have a true cacaphony of color and sound taking place.

Further, the warriors themselves are well-defined, and the animation that occurs when they are in combat, teleporting and re-materializing is excellent. Finally, the chips are laid out in such a way that the overall effect, when surrounded by the hundreds of minute, twinkling stars and the four power pyramids, creates a three-dimensional illusion that works for the viewer. The variety of strategies and hand-eye coordination required to be victorious in combat make Epyx's **Silicon Warrior** well worth the price.□

Some program listings reproduced in **A.N.A.L.O.G.** may contain "strange" characters not shown on the ATARI keyboard. These are special characters which use the CTRL, ESC and "ATARI LOGO" (INVERSE) keys. Shown below is a list of these characters and the keystrokes used to get them. □

```
--- CTRL ,
--- CTRL A
--- CTRL B
--- CTRL C
--- CTRL D
--- CTRL E
--- CTRL F
--- CTRL G
--- CTRL H
--- CTRL I
--- CTRL J
--- CTRL K
--- CTRL L
--- CTRL M
--- CTRL N
--- CTRL O
--- CTRL P
--- CTRL Q
--- CTRL R
--- CTRL S
--- CTRL T
--- CTRL U
--- CTRL V
--- CTRL W
--- CTRL X
--- CTRL Y
--- CTRL Z
--- ESC ESC
--- ESC CTRL UP-ARROW
--- ESC CTRL DOWN-ARROW
--- ESC CTRL LEFT-ARROW
--- ESC CTRL RIGHT-ARROW
--- CTRL .
--- CTRL ;
--- ESC SHIFT CLEAR
--- ESC BACK S
--- ESC TAB
--- INVERSE CTRL ,
--- INVERSE CTRL A
--- INVERSE CTRL B
--- INVERSE CTRL C
--- INVERSE CTRL D
--- INVERSE CTRL E
--- INVERSE CTRL F
--- INVERSE CTRL G
--- INVERSE CTRL H
--- INVERSE CTRL I
--- INVERSE CTRL J
--- INVERSE CTRL K
--- INVERSE CTRL L
--- INVERSE CTRL M
--- INVERSE CTRL N
--- INVERSE CTRL O
--- INVERSE CTRL P
--- INVERSE CTRL Q
--- INVERSE CTRL R
--- INVERSE CTRL S
--- INVERSE CTRL T
--- INVERSE CTRL U
--- INVERSE CTRL V
--- INVERSE CTRL W
--- INVERSE CTRL X
--- INVERSE CTRL Y
--- INVERSE CTRL Z
--- ESC DELETE
--- ESC INSERT
--- ESC CTRL TAB (CLR)
--- ESC SHIFT TAB (SET)
--- INVERSE SPACE
--- INVERSE _
--- INVERSE CTRL .
--- INVERSE CTRL ;
--- INVERSE |
--- ESC CTRL 2
--- ESC CTRL BACK S
--- ESC CTRL INSERT
```

MS. PACMAN
ATARI, Inc.
P.O. Box 61657
Sunnyvale, California 94086
16K Cartridge $39.95

by Kyle Peacock

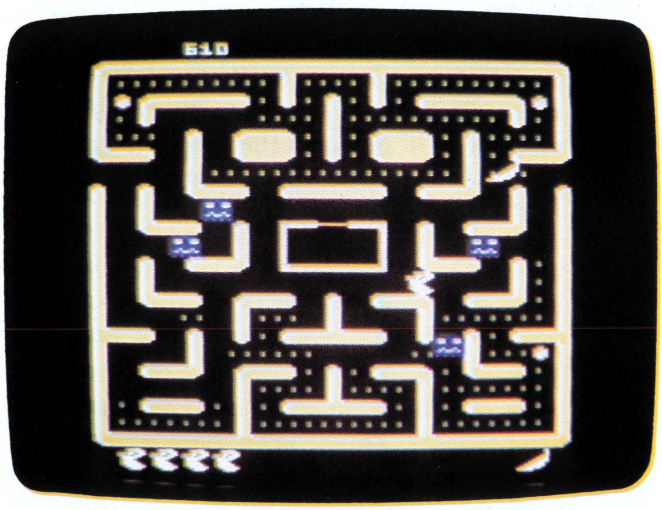
She's here! **Ms. PacMan** at last makes her debut in the home computer market. Naturally, she wants to leave a good impression, so she's lengthened her eyelashes, glopped on some lip gloss and gone to the hairdresser.

Like her legendary predecessor, the object of **Ms. PacMan** is to win points by gobbling up dots in a maze of corridors. Finish all the dots and the maze is refilled for another round. Sound too easy? It would be, if the game didn't include a certain infamous foursome of rag-tag ghosts. Inky, Blinky, Pinky and Sue (Clyde's sister, perhaps?) are back, and they're out to settle the score with **PacMan**'s lady friend.

Should **Ms. PacMan** come in contact with any member of this fearless quartet, she's off to the hurt locker. Fortunately, the lady Pac has a way to turn the tables on the unfriendly ghouls. Digesting one of the four glittering "power pellets" in the maze turns the ghosts into bashful wimps and our heroine into Supergirl, tallying up extra calories and bonus points by devouring her enemies.

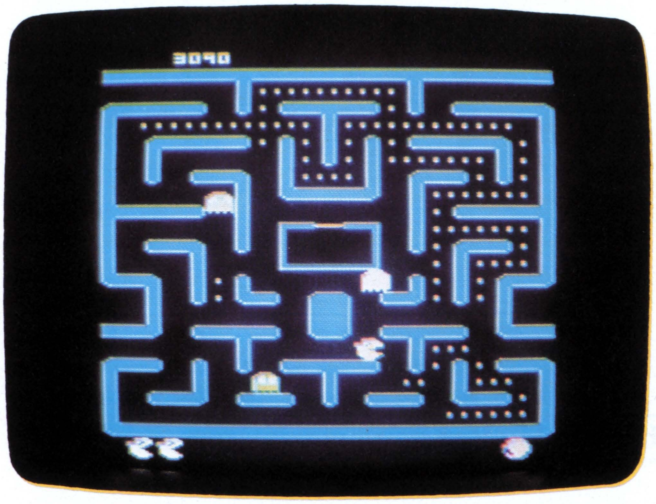
Floating fruit.

Pieces of fruit occasionally wander into the maze through a side door. These valuable prizes randomly prance around, waiting to be gobbled up. The fruit is a diabolical twist on the original **PacMan** concept; often the tempting prizes are just enough of a distraction for the ghosts to get the jump on you. A strategy hint: Fruit always enters or exits the maze through a side door. Hanging around these areas is a fast way to rack up bonus points.



Ms. Pac Man.

Another unique feature of **Ms. PacMan** is the changing mazes. After two levels of a simple pink maze, the game switches to a more complicated blue maze, and so on. High-level mazes require new strategies, as there are more traps and dead-ends to get caught in. Unlike many games, which simply move faster as play progresses, **Ms. PacMan** offers both quick-paced action and changing scenery.



I'm happy to report that Atari has included all of the amusing "intermission" scenes of the arcade classic in this home computer version. They've even gone two steps further by giving you five lives instead of the usual three, and by letting you skip the beginning levels and start at the maze that best suits your capabilities.

Our friend flicker.

Overall, I enjoy **Ms. PacMan**. Both the arcade and home editions offer more challenge and variety than her boring boyfriend. That doesn't mean I'd nominate **Ms. PacMan** for Atari Computer Game of the Year, though. Why not? Well, I have this habit of looking at video games from a programmer's point of view. Does Atari think we haven't seen their **PacMan** for the 2600? Wrong! I've seen it, and made the same comment as everybody else: "Why do the ghosts flicker so badly?" You'll find yourself asking the same question of this computer version of **Ms. PacMan**. Believe it or not, Atari has once again taken the easy way out by multi-tasking the player/missile graphics.

Let's be realistic, guys! Apple and Commodore owners are playing games a lot more complicated than **Ms. PacMan** without any flicker at all. The Atari computer version of regular old **PacMan** doesn't glitch, and neither should its descendants. Too bad an otherwise nice piece of software has to fall prey to lazy programming. If you can ignore the mediocre graphics, **Ms. PacMan** still makes for an enjoyable date, whatever your gender. ☐

BASIC TRAINING

by Tom Hudson

Since the last two installments of **BASIC Training** have been devoted to the IF/THEN structure, I thought it would be good to continue our discussion of control structures. This issue, we'll take a look at one of Atari BASIC's loop control structures, FOR/NEXT.

What will they think of NEXT?

The FOR/NEXT structure is incredibly versatile. Its primary use is for controlling the number of times a loop is executed. A loop is simply a set of instructions executed over and over again. Generally, a loop is executed until some condition is met which allows the program to continue.

FOR/NEXT loops are also used for timing various functions in BASIC. If your program operates too fast, a simple time-delay loop can slow it down to the speed you like. This is often the case with sound generation.

Let's look at the syntax of the FOR/NEXT structure:

FOR avar = aexp1 TO aexp2 (STEP aexp3)
NEXT avar

As you can see, a FOR/NEXT has two parts: The FOR-TO initialization statement with an optional STEP parameter, and the NEXT statement, which ends the loop.

In the above example, "avar" means that any variable name can be used in the indicated position. "aexp" indicates that any valid expression can be used where indicated.

FOR/NEXT actually performs a rather simple job. Look at the program in **Figure 1**.

```
10 FOR COUNT=1 TO 10 STEP 1
20 PRINT COUNT
30 NEXT COUNT
```

Figure 1.

Figure 1 shows a simple, three-line program which will PRINT the numbers from 1 to 10.

Line 10 tells the computer to use the variable COUNT in the FOR/NEXT loop. It will initialize COUNT to 1 and stop when COUNT equals 10. Each time the computer reaches the "NEXT COUNT" statement, it will increment COUNT by 1, indicated by the optional "STEP" value. If the STEP value is not given, the computer will assume a STEP of 1.

Line 20 prints the value of COUNT on the screen.

Line 30 uses the NEXT statement to increment the COUNT variable by the number indicated in STEP, and automatically loops back to the next statement after line 10. In this program, the computer will perform the loop ten times, printing COUNT each time. The result will look like **Figure 2**.

```
1
2
3
4
5
6
7
8
9
10

READY
```

Figure 2.

You can simulate a FOR/NEXT loop very easily. The program in **Figure 3** does the same thing as **Figure 1**, without using the FOR/NEXT structure.

```
10 COUNT=1
20 PRINT COUNT
30 COUNT=COUNT+1
40 IF COUNT<=10 THEN 20
```

Figure 3.

You can see that the function performed by the FOR/NEXT loop is very simple, and easily duplicated by only one extra BASIC statement. Why is it better to use FOR/NEXT?

First, the FOR/NEXT version of the program uses less memory. The code in **Figure 3** requires 25 more bytes than **Figure 1**. This may seem insignificant, but these few wasted bytes can add up to a staggering number in a large program.

Second, the FOR/NEXT version is noticably faster than the equivalent BASIC code. Remember, BASIC interprets each line of code each time it is executed. The extra lines require more time to process than a simple NEXT statement.

Finally, the FOR/NEXT code is much easier to type and easier for other programmers to understand than the equivalent BASIC code. Remember, you may not be the only person who ever looks at your programs, and making the program as readable as possible is very important.

Leaving the loop.

There are two ways to leave a FOR/NEXT loop. The first requires no programmer action, since it is handled by the computer. This occurs when the value of the FOR/NEXT variable exceeds the high value specified by the "TO" parameter. The program in **Figure 4** shows the normal termination of a FOR/NEXT loop.

```
10 FOR X=1 TO 20 STEP 3
20 PRINT "INSIDE LOOP:";X
30 NEXT X
40 PRINT "VALUE AFTER LOOP:";X

RUN

INSIDE LOOP:1
INSIDE LOOP:4
INSIDE LOOP:7
INSIDE LOOP:10
INSIDE LOOP:13
INSIDE LOOP:16
INSIDE LOOP:19
VALUE AFTER LOOP:22

READY
```

Figure 4.

This loop has a STEP value of 3, and, unlike the previous examples, does not end exactly at the "TO" parameter value. You can see that the variable X is incremented by 3 each time the loop is executed, and, as soon as it exceeds the "TO" value of 20, the loop terminates. Note that after the loop ends, the value of the variable is ALWAYS greater than the "TO" value.

The second way to leave a FOR/NEXT loop is with a GOTO statement, as shown in **Figure 5**.

```
10 FOR NUM=11 TO 5 STEP -1
20 PRINT NUM,NUM/2
30 IF NUM/2<5 THEN GOTO 50
40 NEXT NUM
50 END

RUN

11          5.5
10          5
9           4.5

READY
```

Figure 5.

No special programming is needed to terminate FOR/NEXT loops when the GOTO exit is used, but you should be very sure of your program's flow to avoid any problems. One potential problem is running into the NEXT statement after your loop is supposed to be ended. If this happens, the loop may be re-entered, with nasty (and aggravating) results.

Note that in **Figure 5**, the FOR/NEXT loop goes from a high value (11) to a lower value (5). When this structure is needed, a *negative* STEP value is needed for proper execution of the loop.

There is a third possibility for the exiting of a FOR/NEXT loop: *no* exit. How is this possible? **Figures 6** and **7** show two possibilities.

```
10 FOR EVER=1 TO 2 STEP 0
20 NEXT EVER
```

Figure 6.

Figure 6 won't terminate because it has a zero STEP value. It will keep adding 0 to the variable EVER, and will obviously never reach the TO value of 2.

```
10 FOR EVER=1 TO 2
20 EVER=0
30 NEXT EVER
```

Figure 7.

Figure 7 would normally terminate properly, but the loop variable EVER is being reset to zero inside the loop. When the NEXT statement increments the loop variable, it never gets higher than 1. This prevents the loop from terminating.

Leaving the nest.

FOR/NEXT alone is a pretty powerful structure, but you can use several FOR/NEXT loops with what is known as NESTING for complex control loops.

Nesting is simply the act of performing one FOR/-NEXT loop inside of another. **Figure 8** is a short demonstration of how nesting works.

(continued on next page)

```
10 FOR L1=1 TO 5
20 PRINT "LOOP 1:";L1
30 FOR L2=1 TO 5
40 PRINT "LOOP 2:";L2
50 NEXT L2
60 FOR L3=1 TO 5
70 PRINT "LOOP 3:";L3
80 NEXT L3
90 NEXT L1
```

Figure 8.

This program has three loops, two of them inside the third. The lines to the left of the listing show the loop control flow. Note that these control lines do not intersect. You can build loop structures as complex as you like, but you must remember to terminate the inner loop *before* the outer loop. **Figure 9** shows an example of an improperly structured nested loop.

```
10 FOR X=1 TO 10
20 FOR Y=2 TO 20 STEP 2
30 PRINT X,Y
40 NEXT X
50 NEXT Y
```

Figure 9.

You can see by looking at the control flow lines that **Figure 9** violates the non-intersecting control rule. If executed, this program will give an ERROR-13 (no matching FOR statement) at Line 50. This is because the system was confused by the improper FOR/NEXT structure.

Figure 10 shows a useful routine which will clear a two-dimensional numeric array to all zeros.

```
10 DIM ARRAY(10,20)
20 FOR X=0 TO 10
30 FOR Y=0 TO 20
40 ARRAY(X,Y)=0
50 NEXT Y
60 NEXT X
```

Figure 10.

When an array is DIMensioned, it may contain "garbage" and cause problems later in the program. For this reason, all arrays should be zeroed before use. The program in **Figure 10** does this in a fast and orderly manner by using two nested loops. If you print the X and Y values inside the Y loop, you'll see that they go from 0,0 to 10,20. Of course, the loops could be written to zero the array in reverse (FOR X=10 TO 0 STEP -1, FOR Y=20 TO 0 STEP -1) with the same results.

How about a useful example?

Here's a program that illustrates the principles discussed and is also a nice little utility. **Figure 11** is a handy-dandy hexadecimal to decimal converter program. It will convert any base 16 number to base 10. The base 16 number can be up to 9 digits long. Simply enter the hex value when prompted, and the program will print its decimal equivalent. If the number you enter is not a valid hex value, your entry will be displayed with the offending character highlighted in inverse video.

```
10 REM ****************************
20 REM * HEX-TO-DECIMAL CONVERTER *
30 REM *       BASIC TRAINING     *
40 REM *   ANALOG COMPUTING #17   *
50 REM ****************************
60 DIM HEX$(16),IN$(10)
70 HEX$="0123456789ABCDEF"
80 ? :? "ENTER HEX NUMBER";:INPUT IN$:
IF IN$="" OR LEN(IN$)=10 THEN 80
90 DEC=0
100 FOR CHAR=1 TO LEN(IN$)
110 FOR LOOKUP=1 TO 16
120 IF IN$(CHAR,CHAR)=HEX$(LOOKUP,LOOK
UP) THEN 160
130 NEXT LOOKUP
140 TRAP 150:IN$(CHAR,CHAR)=CHR$(ASC(I
N$(CHAR,CHAR))+128)
150 ? "INPUT ERROR - ";IN$:TRAP 40000
:GOTO 80
160 DEC=DEC*16+(LOOKUP-1)
170 NEXT CHAR
180 ? "DECIMAL VALUE:";DEC:GOTO 80
```

CHECKSUM DATA
(see p. 15)

```
10 DATA 335,701,824,204,343,690,126,79
4,121,293,77,491,389,145,519,6052
160 DATA 460,182,682,1324
```

Figure 11.

Line 60 DIMensions the two strings used by the program. IN$ is used to hold the hex value entered by the user. HEX$ is used to hold the 16 hexadecimal digits. This is used as a lookup table for the conversion.

Line 70 places the hexadecimal digits 0-F into HEX$.

Line 80 accepts a hexadecimal number from the keyboard, places it into IN$, and checks its length. If the string is empty or its length is greater than nine digits, the program returns to Line 80 to accept another value.

Line 90 sets the variable DEC to zero. When the conversion is complete, DEC will contain the decimal equivalent of the hex number entered.

Line 100 initializes a FOR/NEXT loop which ranges to Line 170. This is the outer loop of a two-loop nest. As you can see, its loop value is from 1 to the number of characters in IN$, with the default STEP value of 1. Each time this loop is executed, CHAR will point to the next character in IN$, allowing each character to be examined in sequence.

Line 110 starts the second FOR/NEXT loop, the inner loop of the two-loop nest. This loop assigns the variable LOOKUP the values from 1 to 16, with the default STEP of 1. LOOKUP will be used to point to successive characters in HEX$ to see if they match the individual characters of IN$.

Line 120 compares the character of IN$ indicated by CHAR to the character of HEX$ indicated by the variable LOOKUP. If the char-

acters match, the program exits the inner loop with a GOTO instruction to Line 160. If no match is found, the program continues on to the next line.

Line 130 increments LOOKUP to the next character of HEX$. If there are more characters, the program will loop back to Line 120 to continue the hex digit comparison. If all hex digits have been checked, LOOKUP will exceed 16 and the program will continue at Line 140, the error routine.

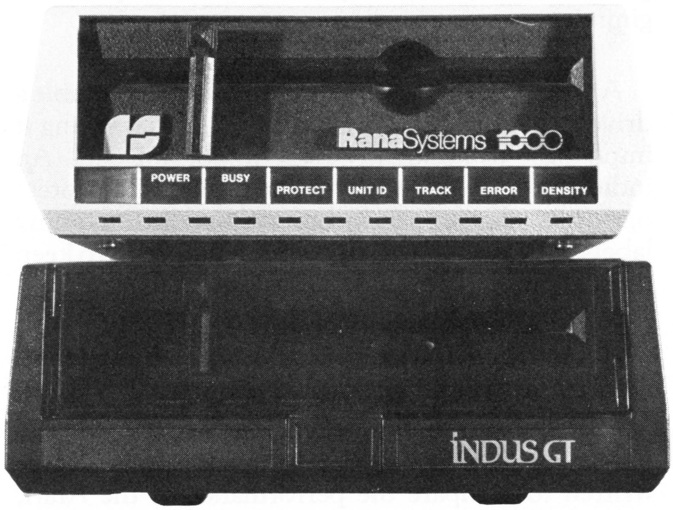
Line 140 is executed if the character entered was not found in HEX$. Let's assume the letter "W" was entered. This is obviously not a valid hexadecimal digit, and would not be found in HEX$. When this happens, Line 140 adds 128 to the invalid character's ATASCII value, changing it to inverse video. If the character is already inverse, the TRAP statement avoids any BASIC error messages by continuing at Line 150.

Line 150 prints the error message and the value entered by the user, with the invalid character shown in inverse. It then turns off the TRAP statement started in Line 140 and exits the FOR/NEXT loop with a GOTO 80.

Line 160 is reached when a valid character is found and matched to a character in HEX$. LOOKUP contains the decimal value of the hex digit plus 1. For example, the value zero in HEX$ has a LOOKUP value of one, and the value F has a LOOKUP value of 16. This line multiplies the DEC value by 16 and adds the value of LOOKUP minus one. The multiply by 16 simply allows for the position value of each base 16 digit.

Line 170 increments the CHAR variable, thereby pointing to the next character of IN$. If there are more characters, the program loops back to Line 110 to search for the new character's hex value. If not, all of IN$ has been scanned, and the program continues to Line 180.

Line 180 prints the final value of DEC, the decimal equivalent of IN$. After printing, the program returns to Line 80, where the user may enter another hex number. □



NEW DISK DRIVES FOR THE ATARI

by Brian Moriarty

Buying a disk drive for your Atari was easy back in the old days. You simply waited until your local dealer got a fresh shipment of 810s, and hoped he wouldn't sell them all before you got there. A few venturesome types might opt for a Percom and indulge in the luxury of double density. But for the majority of users, Atari's lovable old clunker has been the drive of choice since early 1980.

Things are a lot different now. The 810 is history, and retailers are offering all sorts of sexy-looking alternatives, some at very attractive prices. We decided it was time for **ANALOG** to peek at the most popular of the new drives and find out how they stack up in terms of performance and value.

The chart on page 77 compares the basic features of the Atari 1050, Rana 1000, Trak AT-D2 and Indus GT disk drives. Space limitations do not permit a more detailed analysis of so many products; contact the manufacturers at the addresses provided if you need more information.

The following paragraphs are included to help you interpret our comparison data, and to explain how some of the more exotic statistics were obtained.

Density modes.

"Density" refers to the amount of information you can store on a single disk. Three things determine the density of a disk format: the number of bytes in each sector, the number of sectors per track and the number of tracks per disk.

All of the drives in this survey employ a 40-track format. *Single-density* disks have eighteen 128-byte sectors per track, for a total capacity of 18*128*40 or 92,160 bytes. *Double-density* uses exactly the same format, except that the sectors are twice as large (256 bytes apiece). This yields a capacity of 184 kilobytes per disk.

There's third Atari format called *medium density* or *1050 density*, which is supported mainly by Atari's 1050 Disk Drive and DOS 3.0. It's a weird scheme that uses 26 128-byte sectors per track, for a total capacity of approximately 133 kilobytes. The Rana 1000 and Indus GT drives can read and write medium-density disks, but don't expect the 1050 format to become very popular among Atari users.

Built-in diagnostics.

Most of the new Atari drives contain self-diagnostic routines that check out the drive mechanism and/or electronics to make sure they're working okay. The Trak and Indus drives perform these tests automatically every time you turn them on. The Rana 1000 lets you access each test individually.

Track indicator.

A track indictor is a digital readout that informs you which of the 40 disk tracks the drive is currently accessing. I've been using an 810 without a track indicator for three years and never felt any need for

one. Most track indicators are also used for error messages and other legitimate purposes, so we'll forgive the manufacturers for this otherwise pointless gimmick.

Write-protect switch.

A write-protect switch lets you manually disable a drive's writing and formatting functions, making it impossible to accidently erase important data. An indicator light reminds you when the write-protect option is enabled. Sure beats those little gummy labels, although they'll still work if you want to keep using them.

Power switch location.

A disk drive's power switch ought to be up front, where you can reach it easily. Period!

Formatting and access time.

The BASIC benchmark program in **Listing 1** was written to compare the performance of the Atari, Rana, Trak and Indus drives. **Listing 2** is the assembly cource code, composed with **MAC/65**. The benchmark was run five times on each disk drive, and the results averaged to yield the figures shown in our table.

The first part of the program finds out how long it takes the drive to write a "raw" (non-DOS) format on a disk. The Atari's jiffy clock is used to precisely time the formatting speed to within 1/60th of a second.

The second part determines how long it takes the drive to access data under "worst-case" conditions. First, the drive is commanded to read sector #1, located on the outermost track of the disk. As soon as the drive acknowledges receipt of sector 1, the system clock is set to zero and the drive is told to fetch sector 720, which is on the innermost track. A timer reading is taken as soon as Sector 720 is received. The clock is then reset to zero, the drive runs back to fetch sector 1 again, and a second reading is taken upon receipt. This back-and-forth cycle is repeated 8 times; the clock readings are added together and divided by 16 to "smooth" the result.

Noise level.

The access-time test made the drive mechanisms work fairly hard, thereby providing a good opportunity to judge the noise performance of the drives. The ratings in our comparison table are relative to the familiar (and very noisy) Atari 810.

Operating systems.

A disk operating system or DOS is a machine-language program that tells your computer how to communicate with the disk drive. Our table gives the name and manufacturer of the DOS supplied with each drive. These DOSes are so new that we haven't had time to test them yet. Look for a full report on new operating systems, disk utilities and drive enhancements in a future issue.

A note on compatibility.

Each of these new drives is designed to be fully compatible with any program that uses standard Atari I/O procedures to access the disk. Assuming that your drive is aligned properly, you should have no trouble reading any disk written on anybody else's drive, regardless of make.

"Copy-protected" disks are another story. Some of the big-shot game companies are relying on undocumented quirks in the old 810 operating system to keep their products secure. Such disks may refuse to boot on a non-810 drive. Don't pound on your innocent drive if this happens! Blame the software publishers, who continue to disregard the existence of "alien" drives at their own peril. The only way to tell whether a given disk is compatible is to boot it up and pray. And don't be afraid to demand your money back if a "protected" program won't load.

You're going to be living with your new disk drive for a long time. Before you plunk down hundreds of dollars, it's a good idea to get your hands on a sample and play around with it for a few minutes. Things to consider include:

Ergonomics. Is it easy to insert disks into the slot? Can you pull them out without running into the bathroom for tweezers? Do the switches provide positive feedback when you push them? Can you tell the indicators apart without squinting?

Documentation. Does the owner's manual make sense? What do they tell you about the disk operating system?

Cosmetics. Does the drive match the decor of your exquisite 17th-century drawing room? Failing that, you should at least make sure there's enough room for the unit on your computer desk.

Support. What will you do if the drive breaks down tomorrow? What will you do if it quits the day after the warranty expires? You're better off asking now than later.

Selecting a new disk drive from so many contenders won't be easy. But once the choice is made and the check clears, you'll wonder how you ever got along without one. □

Listing 1.

```
10 REM * DISK DRIVE BENCHMARKS
20 FOR I=1536 TO 1678
30 READ BYTE:POKE I,BYTE:NEXT I
40 ? "TIME TO FORMAT DISK:"
50 ? USR(1536);" JIFFIES"
60 ? "WORST-CASE ACCESS TIME:"
70 ? USR(1575)/16;" JIFFIES AVERAGE"
80 END
90 REM * MACHINE LANGUAGE DATA
100 DATA 104,169,1,141,1,3,169
110 DATA 33,141,2,3,169,128,141
120 DATA 4,3,169,4,141,5,3
130 DATA 169,0,133,19,133,20,32
140 DATA 83,228,165,20,133,212,165
150 DATA 19,133,213,96,104,169,1
160 DATA 141,1,3,169,82,141,2
170 DATA 3,169,128,141,4,3,169
```

```
180 DATA 4,141,5,3,216,169,0
190 DATA 133,212,133,213,169,7,133
200 DATA 203,169,1,141,10,3,169
210 DATA 0,141,11,3,32,83,228
220 DATA 169,0,133,20,169,208,141
230 DATA 10,3,169,2,141,11,3
240 DATA 32,83,228,165,20,24,101
250 DATA 212,133,212,144,2,230,213
260 DATA 169,0,133,20,141,11,3
270 DATA 169,1,141,10,3,32,83
280 DATA 228,165,20,24,101,212,133
290 DATA 212,144,2,230,213,198,203
300 DATA 16,185,96
```

•

CHECKSUM DATA
(see p. 15)

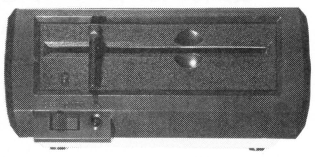
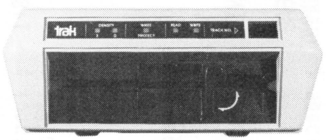
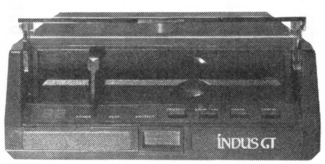
```
10 DATA 126,617,668,643,272,178,539,25
7,57,374,665,864,628,955,518,7361
160 DATA 226,430,371,955,612,164,713,3
53,481,937,376,224,927,957,268,7994
```

•

Listing 2.

```
0100 ; DRIVE BENCHMARKS
0110 ; ----------------
0120 ; Resident disk handler equates
0130 ;
0140 DUNIT = $0301    ; drive #
0150 DCOMND = $0302   ; command register
0160 DBUFLO = $0304   ; lsb of buffer addr
0170 DBUFHI = $0305   ; msb of buffer addr
0180 DAUX1 = $030A    ; lsb of sector #
0190 DAUX2 = $030B    ; msb of sector #
0200 DSKINV = $E453   ; disk handler entry addr
0210 ;
0220 READ =  $52      ; READ SECTOR command
0230 FORMAT = $21     ; FORMAT DISK command
0240 ;
0250 ORIGIN = $0600   ; start of executable code
0260 BUFFER = $0480   ; dummy buffer for sector data
0270 INDEX = $CB      ; loop index register
0280 TICK =  $14      ; lsb of system timer
0290 TOCK =  $13      ; msb of system timer
0300 FR0 =   $D4      ; function return register
0310 ;
0320      *=  ORIGIN
0330 ;
0340 ; TEST #1
0350 ; -------
0360 ;
0370      PLA           ; # arguments; ignore
0380 ;
0390      LDA #1
0400      STA DUNIT     ; specify drive #1
0410 ;
0420      LDA #FORMAT
0430      STA DCOMND    ; specify FORMAT command
0440 ;
0450      LDA # <BUFFER
0460      STA DBUFLO    ; tell the disk handler
0470      LDA # >BUFFER ; where our 128-byte
0480      STA DBUFHI    ; dummy buffer is
0490 ;
0500      LDA #0
0510      STA TOCK
0520      STA TICK      ; zero system timer
0530 ;
0540      JSR DSKINV    ; let OS format disk
0550      LDA TICK
0560      STA FR0       ; put timer reading
0570      LDA TOCK      ; into BASIC's
0580      STA FR0+1     ; function return register
0590      RTS           ; all done
0600 ;
0610 ; TEST #2
0620 ; -------
0630 ;
0640      PLA           ; # arguments; ignore
0650 ;
0660      LDA #1
0670      STA DUNIT     ; specify drive #1
0680 ;
0690      LDA #READ
0700      STA DCOMND    ; specify READ command
0710 ;
0720      LDA # <BUFFER
0730      STA DBUFLO    ; tell the disk handler
0740      LDA # >BUFFER ; where our 128-byte
0750      STA DBUFHI    ; dummy buffer is
0760 ;
0770      CLD           ; for safety
0780      LDA #0
0790      STA FR0
0800      STA FR0+1     ; zero sum
0810      LDA #7
0820      STA INDEX     ; init index
0830 ;
0840 BACK
0850      LDA #1
0860      STA DAUX1
0870      LDA #0
0880      STA DAUX2     ; point to sector 1
0890 ;
0900      JSR DSKINV    ; and fetch it
0910 ;
0920      LDA #0
0930      STA TICK      ; zero timer
0940 ;
0950      LDA # <720
0960      STA DAUX1
0970      LDA # >720    ; now point to
0980      STA DAUX2     ; sector 720
0990 ;
1000      JSR DSKINV    ; and fetch that
1010 ;
1020      LDA TICK      ; get reading
1030      CLC
1040      ADC FR0       ; add to previous
1050      STA FR0       ; readings
1060      BCC FORTH
1070      INC FR0+1
1080 ;
1090 FORTH
1100 ;
1110      LDA #0
1120      STA TICK      ; zero timer
1130 ;
1140      STA DAUX2
1150      LDA #1
1160      STA DAUX1     ; point to sector 1
1170 ;
1180      JSR DSKINV    ; fetch it
1190 ;
1200      LDA TICK      ; get new time
1210      CLC           ; add to previous
1220      ADC FR0       ; readings
1230      STA FR0
1240      BCC NEXT
1250      INC FR0+1
1260 ;
1270 NEXT
1280      DEC INDEX     ; loop 8 times
1290      BPL BACK
1300      RTS           ; and return to BASIC
1310 ;
1320      .END
```

•

DRIVE MODEL	SINGLE DENSITY (92K)?	MEDIUM DENSITY (133K)?	DOUBLE DENSITY (184K)?	BUILT-IN DIAGNOSTICS?	TRACK INDICATOR?	WRITE-PROTECT SWITCH?	POWER SWITCH LOCATION?	NOISE LEVEL?	FORMATTING TIME (JIFFIES)	WORST-CASE ACCESS TIME (JIFFIES)	SUPPLIED OPERATING SYSTEM	SPECIAL FEATURES	DIMENSIONS (HxWxD, INCHES)	MANUFACTURER'S WARRANTY	MANUFACTURER'S LIST PRICE
 ATARI 1050	YES	YES	NO	NO	NO	NO	FRONT	MEDIUM	2015	62.6	Atari DOS 3.0	Matches cosmetics of Atari XL computers.	3½ x 7¼ x 12	90 DAYS	449
 RANA 1000	YES	YES	YES	YES	YES	YES	BACK	LOW	2024	36.9	Rana Smart-DOS	Independent disk formatting. User-programmable diagnostics.	2¾ x 7 x 10¼	120 DAYS	449
 TRAK AT-D2	YES	NO	YES	YES	YES	YES	FRONT	MEDIUM	1094	25.0	OSS DOS XL 2.2	Built-in parallel printer interface (cable additional). 2K printer buffer (expandable). Utilities and game software. Optional "Turbo Software" on EPROM ($60 additional).	3½ x 8 x 11¾	90 DAYS	499
 INDUS GT	YES	YES	YES	YES	YES	YES	BACK	LOW	1354	56.4	OSS DOS XL 2.2	"Beeping" function switches. Word processing, spread-sheet and database software. Hinged dust cover. Hard carrying case.	2¾ x 7 x 11	1 YEAR	449

ATARI, Inc.
P.O. Box 61657
Sunnyvale, California 94086
Tested #59109-493

TRAK MICROCOMPUTER
1511 Ogden Avenue
Downers Grove, Illinois 60515
Tested #32295

RANA SYSTEMS
21300 Superior Street
Chatsworth, California 91311
Tested #ATH-03993

INDUS SYSTEMS
9304 Deering Avenue
Chatsworth, California 91311
Tested #109ARBDO3

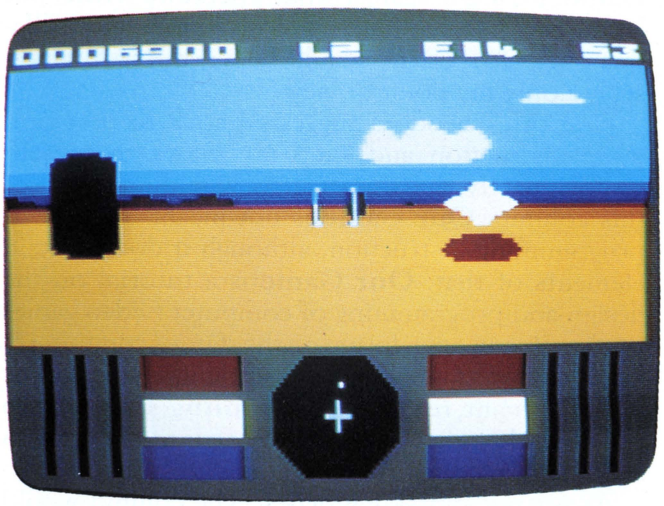
ENCOUNTER!
by Paul Woakes
SYNAPSE SOFTWARE
5221 Central Avenue
Richmond, California 94804
32K Disk $34.95

by Carl Firman

Come with me for a moment. Let's step across the threshold to a galaxy far, far away. . .

Your starship is in orbit around a strange new world. This one wasn't even on the charts. It's ancient — billions of years old. Aside from a few carnivorous plants, no life forms have been detected. So how to explain the hundreds of strange round pillars, spread across the plains in a vast grid? They're obviously artificial, and vaguely reminiscent of the obelisks on Easter Island back on Terra. You're about to find out what they are, because you've just been selected to go planetside with the exploration team.

Your assignment is to pilot one of your starship's antigrav Seekers and defend the explorers. Naturally, your presence is only precautionary. The Treaty of 2076 specifically prohibits the use of destructive weapons unless attacked. That same agreement limits your arsenal to manual phasers and standard deflector shields — no computer-controlled weapons allowed.



The landing vehicle slips silently out of the hanger bay, glides towards the planet's surface and lands on the edge of the flatland. Through the viewpoint you can see some of the strange pillars against the distant horizon. Your job is to survey this area before the exploration team can disembark.

It feels good to be back at the controls of a Seeker. She's a tidy little craft. Smooth, efficient. . .and potentially very deadly. You approach and pass the first of the pillars. Suddenly your scanner shows a blip. You are not alone!

Is it one of the other Seekers? It shouldn't be, and it isn't. A yellow saucer whizzes past your viewport. Your blue "Attack" indicator flashes on, and a bolt of raw energy smashes into your craft. The viewpoint sizzles as the shields absorb the force of the strike. You pull back violently on the stick, only to find yourself surrounded by the menacing pillars! Only one course of action can save you now.

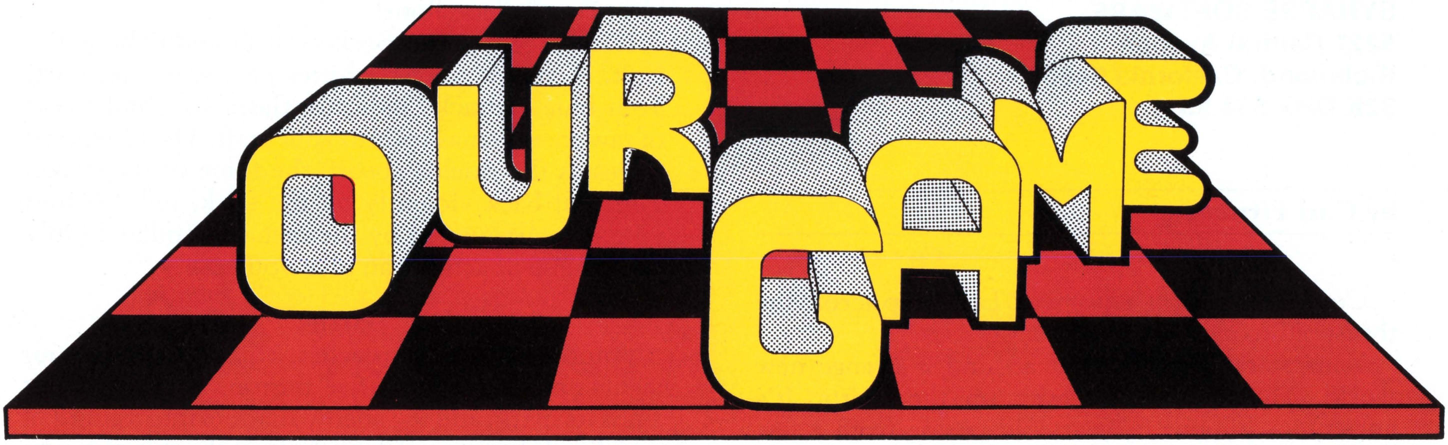
Battle!

Another saucer approaches. You line it up in your sights and squeeze the fire button. A phaser blast finds its mark, and the enemy shatters into a cloud of whirling fragments. *Whew!* At least the things can be destroyed. But these saucers aren't always that easy to kill. They weave and sway like drunken Cygian bouncers. You nevertheless manage to pick off several more, duckling behind pillars to avoid their searing fire.

Your Seeker's red "Missile" indicator has just come on. Sure enough, an alien projectile is screaming straight into your viewport. You line up the sights and fire, but the missile dodges, and you're hit from the side — the rear — and another strike head on! The shield indicator now shows a big fat "0." Another saucer fires — you swerve to evade, but too late. The viewport flashes white and a message appears. *Game Over.* Your hand is sweating as time and space bend back into reality.

You've just had your first **Encounter!** with a military obstacle course — a diabolical machine conceived by a race of beings extinct for half a million centuries. This electronic training ground was built to sharpen the skills of their greatest warriors. Press the START key on your Atari, and you'll find yourself once again thrown into the thick of combat, your battle reflexes locked in a nerve-wracking test to destruction.

The graphics and sound in this 3 dimensional simulation are stunning. There are eight training levels, each with a differently colored landscape and better alien attack strategies. After clearing out all of the saucers and missiles on one level, you must travel through an extradimensional "gate" to reach the next level of difficulty. Avoid those energy spheres flashing past your Seeker, or you'll find yourself back in the previous level. And if you happen to notice another Seeker slugging it out in the grid. . .please, don't shoot me. □



by Joel Gluck

Yes, once again, it's time for **Our Game**, the column that brings you more of what you read **ANALOG** for! In this edition: Viewer Mail, another look at the components of good games, and Debugification!

For those of you who are continuing readers of **Our Game**, you may notice that we still haven't started writing "our game" — the game composed of your mailed-in ideas. Purely to avoid being sent hate mail or dead flowers, let me assure you that development of our game will definitely begin in the next **Our Game** column. I promise!

In the meantime, please continue to send me mail. Reader response so far has not been overwhelming, so I'm still in need of more good, usable ideas.

Viewer mail.

I received this letter from Theodora Brown of New York City:

Since you asked for comments, I assume you're prepared for some negative ones. Personally, I'd prefer if you'd stick to programming instruction and keep your biases to yourself. You are dead wrong on a number of points:

People having the most fun in the arcades are not usually with other people. Sometimes they are, and sometimes they aren't; there is no correlation. And even if they are, so what?

Game manufacturers are not stupid. They know a whole lot more about their market than you do. They are not "hurting themselves" by focusing on male teenagers. If there were a demand for programs for senior citizens, they'd write programs for senior citizens quick enough.

Violence is bad only when it acts on people or things not designed for it. Hitting a baseball is violent, but it's not harmful. Get the difference? Do you think that hitting a baseball encourages a youngster to go home and hit a vase or his baby brother with the bat? If hitting a baseball is harmless, then zapping a Zylon is also harmless. Where did this anti-violence kick come from anyway? It's quite boring.

I'm glad that somebody finally wrote a letter that is at odds with some of the points I've made in **Our Game**. I'll argue against what Theodora wrote, but let me say first that I hope I receive more letters like this one. **Our Game** would be worthless if it presented only my point of view.

First off, Theodora suggests I stick to programming instruction and keep my biases to myself. Unfortunately, that does not jive with the purpose of this column. **Our Game** is not meant to serve as programming instruction, although it does contain elements of that. **Our Game** is a tutorial on the design and programming of computer/video games — as well as an open forum on the state of computer/-video games. If straight programming instruction is what you want, read **BASIC Training**, **Boot Camp** or any of the other excellent tutorials to be found in **ANALOG Computing**.

As for my expressing biases: The column would probably not be very interesting if I kept my opinions to myself. Much of what I write is designed to elicit reader response. The mere fact that Theodora disagreed with my opinions and wrote a letter means that what I am doing is working.

I disagree with Theodora's point that there is no correlation between having more fun in arcades and being with other people. My opinion is based on my observations in numerous arcades and game rooms,

and my own personal experiences. Man is a social animal, despite what some computer or video game junkies may think. Also, I think that Theodora's point that it doesn't matter either way is also wrong: Only when game designers and manufacturers realize the role of the human interaction that takes place in arcades (and at home) will we see better multiple-player games that are truly interactive. The importance of having games like this is clear; interactive games help teach social and emotional lessons, whereas single-player games tend to be far less beneficial. Not everyone worries about someone who has few friends and spends too much time playing video games — but it's an important issue nonetheless.

Theodora also pointed out that there is no demand for game programs for senior citizens. I agree. However, I picture things the other way around. From my point of view, most game manufacturers have never seriously tried to promote games that appeal to different age groups. Senior citizens may be an extreme example, but certainly an effort can be made to target games to young children or to adults — not games that are mere variants of the standard arcade chase'em-down-shoot'em-up, but rather games with social and educational value.

Not that everyone is ignoring the issue. Atari is one company that has made some effort along these lines. Working with the Children's Television Workshop, Steve Wright of Atari helped produce a batch of excellent educational young children's games for the Atari 2600.

My point is that it does not matter whether or not there is a demand for games of this sort — it is up to game manufacturers to create the demand. All it takes is creativity and the will to pioneer new markets.

As for the question of violence in video games: I suspect that a violent game's effect on a person (especially a child) is similar to the effects of television violence. The two are comparable because a video game's lack of realism is compensated for by its being interactive. Unlike television, in a video game *you* are the one who is killing or being killed. It is difficult, however, to prove that violent video games are more harmful than nonviolent ones.

But I have a better reason for avoiding violent games. They are trite. No theme has been more overused in the realm of video games than kill-or-be-killed, especially in the form of what is ordinarily called the "shoot'em-up." It is in this sort of game that we see the game manufacturers appealing to the lowest common denominator. There are rarely improvements in the game ideas themselves, but only in the technology of the implementation. Atari's **Star Wars** is a fine example; behind all of its high-tech trappings, the game is essentially just another shoot'em-up.

So, this "anti-violence kick" that Theodora complains about is my reaction to the lack of creativity on the part of game designers and manufacturers. In our game, we will try to avoid hackneyed themes and search for new and better ones. Why? Because it's a challenge.

I also received this letter from Eric Hansotte of Glenshaw, Pennsylvania:

I believe that the right kind of board game can entertain almost anyone and never have him/her become sick or frustrated with it. . .Just compare chess, checkers, and **Monopoly** *to* **PacMan**! *No contest!*

So how about a board-type computer game that incorporates the sound, graphics, and other game-playing capabilities of the computer as an added dimension. This might be a game in which the "board" is the main screen. Players advance around a pattern by some cleverly determined method of movement as they accumulate points, money, awards, or whatever. At certain points throughout the board pattern there should be "stations," similar to those in a regular board game, that might ask you to pick a card. Here, at these stations, is where we might have the sound and graphics come into play, as the computer would go to a different graphics mode and let the player perform a task. . . The task would have to be something interesting, the outcome of which would be good or bad for the player, such as an award or being sent back to "start."

Great idea, Eric. Before I even learned BASIC, one of my favorite pastimes was designing my own board games, using magic markers and oak-tag paper, and cut-up file cards for the playing pieces. I've always thought it would be great to do a computer/board game — and it seems you've figured out a good way to do it. This is a definite possibility for our game.

By the way, one of the most interesting games of the past year for the Atari computer was a computer/-board game called **Archon**, published by Electronic Arts. Player interaction in the game is beautifully done (although there is also a one-player option), and the game itself is imaginative and fun. I recommend it to anyone looking for something a little different.

Barbara Baker writes from San Bernardino, California:

I agree that violence in games does not appeal to most women, but I think people of either sex are enchanted by the idea of being a hero. Here is my game idea:

Underground Railroad. *The object is to lead a group of slaves from the Deep South to freedom in the North. The size of the group is your choice: large groups gain more points, but small ones are easier to handle without losing anyone in your party. The hazards are environmental, such as snake-filled swamps and ice-packed rivers.*

There are also farms along the way where you can sometimes obtain food, because a well-fed party travels faster. All along you must evade the pursuing bloodhounds by various tricks such as doubling back or crossing water.

The concept might fit better in a science-fiction setting, where you can invent more imaginative hazards. However, I think that presenting a heroic act without violence is the most broadly-appealing way to go.

Barbara's game idea is one of the most unique and creative I've received. But one must keep in mind the limitations we have in terms of speed and memory: it would be very difficult to implement a game as complex as **Underground Railroad** on the Atari computer in BASIC. Also, when developing a game idea, it helps to take it past the storyline and stage and consider the graphic representation. How would we represent (not to mention control) a group of people moving around on the screen? How would obtaining food from a farm work graphically, for example? All of these questions must be answered before an implementation can be attempted.

That's all for viewer mail. Keep those letters coming!

Good games, revisited.

A while back we talked about good games. What makes a good game?

The first motive of any game design should be to entertain. A game is not a game if it is not fun. After that, a game designer may have other motives:

1) **To educate**. There is a sad lack of good educational software on the home computer market. Much of what has been produced is boring or poorly designed. Approaching the educational market from the direction of entertainment instead of education may be the best way to go.

2) **To simulate**. Simulations can be enjoyable if designed correctly. They can be difficult to write, given the constraints of a microcomputer, unless what one is simulating can be simplified. Simulations that are accurate can be educational as well. Good examples are Chris Crawford's **Scram** and **Eastern Front 1941**. A common pitfall, though, is to assume that an accurate simulation is automatically fun to use; this is frequently not the case.

3) **To experiment**. A game designer who desires to break free of the common computer/-video game form may find unexpected resistance. Douglas Crockford of Atari wrote a game last summer called **Hollywood Medieval**, an unusual game in which a player uses only one control button (the START key) to make his way through a maze. The maze is not visual,

however — it's composed of music (although Doug did create beautiful 3-D graphics to accompany the action of the game). Unfortunately, Doug's game was not accepted by the Atari Program Exchange, which probably found the game a bit too strange to sell. From my point of view, the purpose of a group like APX is to make available unique games like **Hollywood Medieval**.

Not all experimental games fail, of course. One of last year's best was Bill Budge's **Pinball Construction Set** (Electronic Arts), a program that allows you to design complex video-pinball machines on the screen without having to deal with a complex interface. The idea was good, the implementation was excellent, and the result was unlike any other game.

4) **To make money**. Almost all game writers have this motive in mind to some extent. Unfortunately, it conflicts in some ways with the motives mentioned above, especially the desire to experiment; the easiest way to make money in computer games is to play it safe. Stick to old forms, razzle-dazzle'em with better special effects, and take full advantage of advertising and hype — that's the formula for a hit game. Fortunately, one can still be creative and make money, but it may be a struggle.

There are other considerations when one talks of good games. How about *themes?*

My dictionary equates, for the most part, "themes" with "topics." This works pretty well when speaking about video games. The theme of a game like **Galaxians**, for example, could be "outer space adventure."

For many games, though, it is hard to pinpoint a theme. Some seem to have none at all. Take, for example, the prototype-game presented last month, called **Four Letter Words**. **FLW** was an excellent example of a "game-for-games'-sake." It had no identifiable real-world or fictional theme. This is not necessarily a bad thing, but people usually find it easier to learn and enjoy the games that have strong, identifiable themes — games like **Joust**, **Centipede** and **Front Line**. As a final example, one arcade game that seemed devoid of a strong theme was Atari's **Tempest**.

Traditionally, games must have an object. A game without an object is what I like to call a "toy." A simple example of this is the **Bounce** game presented in the January **Our Game** (Issue #15). Toys are just like games except it is not clear that one is supposed to do anything in particular with them. They have no formal goal, like winning or achieving a high score.

Most computer/video games use the *score* as the device for making the object of the game clear. The reason behind this is that most computer/video games are inspired by the arcades, where one plays a game until one loses. There is no goal to reach, no task to complete, only an increasing score and an increasingly difficult game. Recently, though, there have been games like **Pole Position** and **Track & Field**, which break away from the usual format by letting the game end.

In most arcade games, the object is to "kill the enemy for points and avoid being killed." This is true of oldies like **Space Invaders**, **Asteroids** and even **PacMan** (remember, you can eat those goblins), and it's still true today. Of course, this object does not apply to all games: take **Pong** and **Breakout** (or almost anything with a bouncing ball), or most sports games, for example.

In games for home computers, one has much more freedom to design an object that is attainable; the game may end. **FLW** is typical of any of several non-arcade games. There is also the adventure game, something the arcade manufacturers haven't quite figured out how to properly implement in their standard format.

One final element that comes to mind when talking about game design is *self-consistency*. Something is self-consistent when all of its parts properly fit in with the whole. This is especially important to consider when one has a strong theme. If I was writing an action "Jungle Adventure" game, it would not be self-consistent to give the lions and tigers light-sabres to use as weapons. Yes, a dash of silliness like that can add humor to a game, but it can also ruin any illusion of realism (or fantasy) that may have been attempted.

Try to keep all of these elements in mind when mulling over game ideas. The more well thought out your submissions to **Our Game**, the better our game will be.

Debugification.

One of the most useful skills that frequent programming will give you is the ability to debug (find and correct errors) quickly and efficiently. I say useful because debugging is more than just something to do when you program — it is a way to approach problems in general. What follows are some hints for effective debugging:

1) **Try to write clear code**. Use remarks. Program in a structured fashion, dividing your task into individual procedures. All of this will help when you are wading back through a program hunting a bug.

2) **Keep it simple**. Do not use optimizing (speed-increasing or memory-saving) tricks unless you really need them. Getting the whole thing to work comes first, optimizing comes later. Also, don't borrow complex routines from other programs without careful fore—thought (How does it work? How does it affect my program?). This, too, will make your program easier to debug.

3) **Debug in an orderly fashion**. Follow a general outline for testing a program or routine. This one works for me:

a) Run program or routine.

b) Write down all percieved errors.

c) Correct fatal errors (errors that cause your program to stop) immediately and re-run.

d) If you make it through the whole thing, correct non-fatal errors in the order that you spotted them.

e) Re-run and test after each major error-correct.

f) If necessary, go through the process again with different input values. Different parameters may point out new errors. Always try out the boundary (highest & lowest) values for inputs.

4) **Beware of sly bugs**. Keep in mind that one bug-fix may create a different bug. Also, two bugs can cancel each other out (just like two mistakes in a math problem). Finally, some bugs only show up under strange situations. Try to cause those situations when debugging.

5) **Help**! Don't be afraid to ask someone more experienced for help with a bug. Clear code (#1) will help her when she looks at your program. Also, don't bias her away from possible solutions by immediately listing things you have already tried.

6) **Trap the bug**. Frequently you can do things that will make the source of an error clear:

a) Try different input or parameter values and compare the results.

b) Insert "checkpoint" PRINTs or even SOUNDs to let you know when the computer has reached a certain part of your program (like PRINT "Here at #1").

c) Replace extraneous statements with "dummy" statements (put a "REM" in front of them, (for example) to isolate the cause of a bug.

d) If you need to take a closer look at what is happening, slow down a routine by adding pause loops (like FOR PAUZ=1 TO 100: NEXT PAUZ).

e) Insert "diagnostic" PRINTs to let you know the values of certain variables in a part of your program (like PRINT A, J5, TNAME$).

7) **Watch yourself**. If you get confused and muddled, re-evaluate what you want to do in that part of the program, organize it on paper, and re-write it. If things get really bad, don't hit the computer — take a break. Also, make sure to get plenty of sleep, food and liquids.

That's all for my debugging hints. If you can think of any more, send them along and I'll pass them on to the readers.

We could send letters.

What? You say you like the idea of **Our Game** but haven't written me a **letter** yet? For shame, for shame!

All you have to do is get a piece of paper, write "Dear Joel" at the top, and write down the best idea for our game that you can think of. Also include anything you'd like to say about computer/video games in general. Then sign your name and put the piece of paper in an envelope. On the outside of the envelope it helps to write:

Our Game
c/o ANALOG Computing Magazine
P.O. Box 23
Worcester, MA 01603

Then just put a stamp on it and a return address, and mail the little bugger! Wasn't that easy? (Don't answer that.)

In next month's **Our Game**, we will definitely start "our game," as well as talk about the ancient Sumerian practice of playtesting: how to do it, where to do it, and who you should do it with. Get excited and stay tuned. □

16K Cassette or 24K Disk

PLANETARY DEFENSE

by Charles Bachand and Tom Hudson

Think you can defend a planet from enemy space-bombs? This issue's assembly-language game will let you find out. **Planetary Defense** is a one-player game that can be played with either a joystick or Koala Technologies Corporation's "KoalaPad."

Something new.

If you're familiar with **ANALOG**'s previous assembly-language games, you'll notice something new about this one: There's only one BASIC listing! Our early efforts included modification listings for cassette and disk versions. This new approach allows you to make either version using just one program, and reduces the possibility of typing errors.

Before typing anything, look at the listings accompanying this article.

Listing 1 is the BASIC data and data checking routine. This listing is used to create both cassette and disk versions of **Planetary Defense**. The data statements are listed in hexadecimal (base 16), so the program will fit in 16K cassette systems. This makes typing more difficult, but it's a necessary evil.

Listing 2 is the assembly-language source code for **Planetary Defense**, created with the OSS **MAC/65** assembler. You do *not* have to type this listing to play the game! It is included for those readers interested in assembly language.

Follow the instructions below to make either a cassette or disk version of **Planetary Defense**.

Cassette instructions.

1. Type **Listing 1** into your computer using the BASIC cartridge, and verify your typing with C:CHECK (see page 15).

(Continued on page 85)

ATARI®

ATARI is a trademark of ATARI, INC.

NEW LOWER PRICES!!

ATARI 800XL	**Call**
Atari 1050 Disk Drive	**Call**
Atari 1010 Recorder	**77.00**
Atari 1027 Printer	**Call**
Atari 850 Interface	**Call**

MOSAIC

64K Ram/400	149.00
64K Ram + Cable Kit/400/800	169.00
48K Ram Kit	94.00
16/32 Expander	64.95
32K Ram	77.95
Mosaic Adaptor	49.95

Memory Expansion for Atari 600XL . . Call

DISK DRIVES

Percom . . . . .
Trak . . . . Call
Rana . . . . For
Indus Gt . . Prices

NEW

RS232 Modem Adaptor (Through serial port) 39.95

Koala Touch Tablet Cart or Disk . . . 69.95

MONITORS

We carry a full line of AMDEK and USI Monitors.

LET YOUR ATARI IMPROVE YOUR LIFE AND YOUR MIND

NEW - THE STIMUTECH SUBLIMINAL PROGRAM

With the **Expando-Vision Interface** the Atari Computer user can now work on improving his or her personal environment during normal television viewing with a proven psychological technique that was banned by the F.C.C.

Buy the Expando-Vision Interface for $99 and get the Rom Cart of your choice for **Free!!!**

Rom Carts are available for:

Weight Control	Drinking Control
Study Habits	Smoking Control
Stress Control	Career Success

Sexual Confidence

Additional Rom Carts $29.95 each

3RD Party Printer Interfaces

Interfast I	139.00
Apeface w/cable included	69.95

PRINTERS

SUPER SPECIAL - Prowriter Package with Apeface or Interfast I . . . . Call

Axiom AT-100	229.00
Axiom AT-550	329.00

ATARI

Programming 2 & 3 - T	23.95
Conversational Lang. - T	44.95
Music Composer - Cart	32.95
My First Alphabet - D	26.95
Touch Typing - T	19.95
Home Filing Mgr. - D	37.95
Mailing List - T	19.95
Caverns of Mars - D	28.95
Computer Chess - Cart	26.95
Missile Command - Cart	26.95
Super Breakout - Cart	26.95
Star Raiders - Cart	32.95
Assembly Editor - Cart	46.95
Basic - Cart	41.95
Macroassembler - D	67.95
Pilot (Home Pkg.) - Cart	58.95
Invitation to Program I T	19.95
Speed Reading - T	55.95
Basketball - Cart	26.95
Graph-it - T	15.95
Juggles House - D/T	22.95
Pilot (Educator) - Cart	97.95
Video Easel - Cart	26.95
Defender - Cart	32.95
Galaxian - Cart	32.95
Qix - Cart	32.95
Dig Dug - Cart	32.95
ET - Cart	37.95
Timewise - D	23.95
Atariwriter - Cart	74.95
Donkey Kong - Cart	37.95
Ms. Pac-Man - Cart	39.95
Tennis - Cart	35.95
Eastern Front - Cart	32.95
Donkey Kong Jr. - Cart	39.95
Pengo - Cart	35.95
Logo - Cart	79.95
Robitron - Cart	35.95
Pole Position - Cart	39.95
Microsoft Basic II - C	67.95
Paint - D	33.95
Caverns of Mars - Cart	32.95
Joust - Cart	39.95
Visicalc - D	159.95

THORN

War Games - Cart	27.95
Hockey - Cart	27.95
Soccer - Cart	30.95
Submarine Commander - Cart	30.95

SIERRA ON-LINE

Homeword - D	Call
Dark Crystal - D	27.95
Frogger - D/T	23.95
Quest For Tires - D	23.95
Ultima I - D	23.95
Ultima II - D	41.94
Wizard/Princess - D	22.95

SPINNAKER

Up For Grabs - Cart	27.95
Cosmic Life - Cart	23.95
Facemaker - Cart/Disk	23.95
Alphabet Zoo - Cart	23.95
Fraction Fever - Cart	23.95
Kids on Keys - Cart	23.95
Delta Drawing - Cart	27.95
Story Machine - Disk	23.95
Rhymes & Riddles - D	20.95
Amazing Things - D	27.95
Hey Diddle Diddle - D	20.95
Snooper Troops 1 or 2 - D	30.95
Trains - D	27.95
Aerobics - D	30.95

SIRIUS

Bandits - D	23.95
Way Out - D	27.95
Repton - D	27.95
Twerps - D	23.95
Blade of Blackpoole - D	27.95
Type Attack - D	27.95
Beer Run - D	23.95
Critical Mass - D	27.95
Wavy Navy - D	23.95

GAMESTAR

Baja Buggies -D/T	22.95
Starleague Baseball - D/T	22.95
Starbowl Football D/T	22.95

SEGA

Buck Rogers - Cart	27.95
Star Trek- Cart	27.95
Congo Bongo- Cart	27.95
Tac Scan- Cart	27.95

EPYX

Gateway To Apshai - Cart	27.95
Pitstop - Cart	27.95
Lunar Outpost - Cart	27.95
Swat Rescue - Cart	27.95
Silicon Warrior - Cart	27.95
Fun With Music - Cart	27.95
Jumpman Jr. - Cart	27.95
Dragonriders of Pern - D/C	27.95
Temple of Apshai - D/C	27.95
Hellfire Warrior - D/C	20.95
Fun With Art - Cart	27.95

INFOCOM

Deadline - D	34.95
Witness - D	34.95
Zork I, II, III - D	27.95
Starcross - D	27.95
Planetfall - D	34.95
Suspended - D	34.95
Enchanter - D	34.95
Infidel - D	34.95

Hundreds of items available for the ATARI . . . please call.

LJK

Letter Perfect - D	74.95
Data Perfect - D	74.95
Spell Perfect - D	64.95

BRODERBUND

AE - D	23.95
Arcade Machine - D	41.95
Bank St. Writer - D	49.95
Lode Runner - D	23.95
Drol - D	23.95
Spare Change - D	23.95
Choplifter - D	23.95

DATA SOFT

Money Wizard - D	49.95
Letter Wizard - D	49.95
Spell Wizard - D	34.95
Genesis - D/T	20.95
Heathcliff - D/T	20.95
Pooyan - D/T	20.95
Zaxxon - D/T	27.95
Micropainter - D	23.95

SYNAPSE

Blue Max - D/T	23.95
Dimension X - D/T	23.95
Drelbs - D/T	23.95
Encounter - D/T	23.95
Necromancer - D/T	23.95
Quasimoto - D/T	23.95
Rainbow Walker - D/T	23.95
Shamus Case II - D/T	23.95
Zeppelin - D/T	23.95
N.Y.C. - D/T	23.95

FIRST STAR

Bristles - D/T	20.95
Flip Flop - D/T	20.95
Astro Chase - D	20.95
Boulder Dash - D/T	20.95

STRATEGIC SIMULATIONS

Carrier Force - D	41.95
Combat Leader - D/T	27.95
Battle For Normandy -D/T	27.95
Eagles - D/T	27.95

PARKER BROS.

Astrochase - Cart	34.95
Frogger - Cart	34.95
Q*Bert - Cart	34.95
Popeye - Cart	34.95

MISCELLANEOUS

Castle-Wolfenstein - D	20.95
Home Accountant - D	52.95
Master Type - D/Cart	27.95
Spelunker - D	27.95
Flight Simulator II - D	37.95
Zombies - D/T	23.95
Mr. Robot - D	23.95
River Raid - Cart	31.95
Diskey - D	34.95
Chatterbee - D	27.95
Rally Speedway - Cart	34.95
Ultima III - D	41.95
Miner 2049'er - Cart	34.95
Scraper Caper - Cart	34.95
Basic XL - Cart	74.95
Monkey Wrench II - Cart	49.95
Omnimon	82.95
Scraper Caper - Cart	34.95

BOOKS

Compute's First Bk/Games	12.95
Inside Atari DOS	19.95
Mapping the Atari	14.95
Machine Language/ Beginners	14.95
Your Atari Computer	16.95
The Atari Assembler	12.95
Visicalc Book-Atari	14.95

ACCESSORIES

WICO Joystick	Call
Flip 'n' File Trunks	20.95
Flip 'n' File Cart	20.95
Joysensor	24.95
Elephant Disks (Box of 10)	20.00
WICO Trakball	37.95
KRAFT Joystick	15.95

D - Disk T - Cassette Cart - Cartridge

ComputAbility™

To Order Call Toll Free

800-558-0003

For Technical Info, Order Inquiries, or for Wisc. Orders -

414-351-2007

NO SURCHARGE FOR MASTERCARD OR VISA

COMPUTABILITY
P.O. Box 17882
Milwaukee, WI 53217

MasterCard VISA

ORDER LINES OPEN
Mon-Fri 11 AM - 7 PM CST
Sat 12 PM - 5 PM CST

ORDERING INFORMATION For fast delivery send cashier's check, money order or direct bank transfers. Personal and company checks allow 2 weeks to clear. Charges for COD are $3.00. School Purchase Orders welcome. In CONTINENTAL USA, include $3.00 shipping per software order. Include 3% shipping on all Hardware orders, minimum $3.00. Mastercard & Visa please include card # and expiration date. WI residents please add 5% sales tax. HI, AK, FPO, APO, Canadian orders - add 5% shipping, minimum $5.00. All other foreign orders, please add 15% shipping, minimum $10.00. All goods are new and include factory warranty. Due to our low prices, all sales are final. All defective returns must have a return authorization number. Please call 414-351-2007 to obtain an RA# or your return will NOT be accepted for replacement or repair. Prices and availability are subject to change without notice.

2. Type RUN and press RETURN. The program will prompt you with:

```
MAKE CASSETTE (0), OR DISK (1)?
```

Type 0 and press return. The program will now begin checking the DATA statements, printing the line number of each as it goes. It will alert you if it finds any problems. Fix any incorrect lines and re-RUN the program if necessary until all errors are eliminated.

3. When all DATA lines are correct, the computer will beep twice and prompt you to READY CASSETTE AND PRESS RETURN. Insert a blank cassette in your recorder, press the RECORD and PLAY buttons simultaneously and hit RETURN. The message WRITING FILE will appear, and the program will create a machine-language boot tape version of **Planetary Defense**, printing each DATA line number as it goes. When the READY prompt appears, the game is recorded and ready to play. CSAVE the BASIC program onto a separate tape before continuing.

4. To play the game, rewind the tape created by the BASIC program to the beginning. Turn your computer OFF and remove all cartridges. Press the PLAY button on your recorder and turn ON your computer while holding down the START key. If you have a 600 or 800 XL computer, you must hold down the START and OPTION keys together when you turn on the power. The computer will "beep" once. Hit the RETURN key and **Planetary Defense** will load and run automatically.

Disk instructions.

1. Type **Listing 1** into your computer using the BASIC cartridge, and verify your typing with D:CHECK2 (see page 15).

2. Type RUN and press RETURN. The program will ask:

```
MAKE CASSETTE (0), OR DISK (1)?
```

Type 1 and press RETURN. The program will begin checking the DATA lines, printing the line number of each statement as it goes. It will alert you if it finds any problems. Fix incorrect lines and re-RUN the program if necessary until all errors are eliminated.

3. When all DATA lines are correct, you will be prompted to INSERT DISK WITH DOS, PRESS RETURN. Put a disk containing DOS 2.OS into drive #1 and press RETURN. The message WRITING FILE will appear and the program will create an AUTORUN.SYS file on the disk, displaying each DATA line number as it goes. When the READY prompt appears, the game is ready to play. Be sure the BASIC program is SAVEd before continuing.

4. To play the game, insert the disk containing the AUTORUN.SYS file into drive #1. Turn your computer OFF, remove all cartridges and turn the computer back ON. **Planetary Defense** will load and run automatically.

Playing the game.

Once started, **Planetary Defense** will display the title screen and controller options.

To use a joystick, plug it into controller jack #1 and press either START or the joystick button to begin the game.

To use the KoalaPad, plug it into controller jack #1 and press either SELECT or one of the KoalaPad buttons to begin the game.

You are in charge of protecting your planet from alien attackers. Your only defense is an orbiting satellite armed with a charged-particle gun. To aim the gun, you simply move the targeting crosshairs to the desired point with the controller. Press your controller button to fire at the target. Once fired, you have no control over the charged-particle projectile. Up to six projectiles may be active at a time. Be careful about where you shoot — if you're not careful, you can hit your own planet and destroy part of it!

Your usual target will be the red space-bombs being dropped by the aliens. These bombs head toward the center of your planet. If they hit the planet they will explode, destroying a chunk of it. If any explosion ever reaches the core of the planet, the game is over. If you destroy a space-bomb before it hits your planet, you are awarded from 10-120 points, depending on the level.

Another threat that is much more dangerous than the space-bombs is a flying saucer that cruises along, firing its own charged-particle weapons at your planet! If not stopped quickly, this alien menace will do an incredible amount of damage. The saucer appears more often at higher levels, so be on the lookout! Saucers are worth from 100-400 points each, depending on the level.

If your satellite ever collides with an incoming space-bomb, the satellite will be destroyed. If the collision is a direct hit, the bomb will also explode, giving you the appropriate amount of points. This is a costly way to gain points, as you have only five satellites. Once all five are destroyed, the game is over, and your planet will be overrun by enemy bombs.

Your score is shown at the upper left of the screen. The number of the level you are on is shown at the upper center of the screen. The number of satellites you have is shown at the upper right by the "*" characters. The space bar pauses the game. Good luck! □

BASIC Listing.

```
10 REM *** PLANETARY DEFENSE ***
20 TRAP 20:? "MAKE CASSETTE (0), OR DI
SK (1)";:INPUT DSK:IF DSK>1 THEN 20
30 TRAP 40000:DATA 0,1,2,3,4,5,6,7,8,9
,0,0,0,0,0,0,0,10,11,12,13,14,15
40 DIM DAT$(91),HEX(22):FOR X=0 TO 22:
READ N:HEX(X)=N:NEXT X:LINE=990:RESTOR
E 1000:TRAP 120:? "CHECKING DATA"
50 LINE=LINE+10:? "LINE:";LINE:READ DA
T$:IF LEN(DAT$)<>90 THEN 220
60 DATLIN=PEEK(183)+PEEK(184)*256:IF D
ATLIN<>LINE THEN ? "LINE ";LINE;" MISS
ING!":END
70 FOR X=1 TO 89 STEP 2:D1=ASC(DAT$(X,
X))-48:D2=ASC(DAT$(X+1,X+1))-48:BYTE=H
EX(D1)*16+HEX(D2)
80 IF PASS=2 THEN PUT #1,BYTE:NEXT X:R
EAD CHKSUM:GOTO 50
90 TOTAL=TOTAL+BYTE:IF TOTAL>999 THEN
TOTAL=TOTAL-1000
100 NEXT X:READ CHKSUM:IF TOTAL=CHKSUM
 THEN 50
110 GOTO 220
120 IF PEEK(195)<>6 THEN 220
130 IF PASS=0 THEN 170
140 IF  NOT DSK THEN 160
150 PUT #1,224:PUT #1,2:PUT #1,225:PUT
 #1,2:PUT #1,18:PUT #1,33:CLOSE #1:END
160 FOR X=1 TO 34:PUT #1,0:NEXT X:CLOS
E #1:END
170 IF  NOT DSK THEN 200
180 ? "INSERT DISK WITH DOS, PRESS RET
URN";:DIM IN$(1):INPUT IN$:OPEN #1,8,0
,"D:AUTORUN.SYS"
190 PUT #1,255:PUT #1,255:PUT #1,0:PUT
 #1,32:PUT #1,181:PUT #1,45:GOTO 210
200 ? "READY CASSETTE AND PRESS RETURN
";:OPEN #1,8,128,"C:":RESTORE 230:FOR
X=1 TO 40:READ N:PUT #1,N:NEXT X
210 ? :? "WRITING FILE":PASS=2:LINE=99
0:RESTORE 1000:TRAP 120:GOTO 50
220 ? "BAD DATA: LINE ";LINE:END
230 DATA 0,28,216,31,255,31,169,0,141,
47,2,169,60,141,2,211,169,0,141,231,2,
133,14,169,56,141,232,2
240 DATA 133,15,169,18,133,10,169,33,1
33,11,24,96
1000 DATA 707070707070707070461C207007
70061006707006200640064100200021 2E212C
2F2700232F2D303534292E27,480
1010 DATA 07330040706C616E657461727940
40646566656E7365408 0A2B980A3A8A1B2ACA5
B380A2A1A3A8A1AEA480C0C0,82
1020 DATA C0E1EEE4C0F4EFEDC0E8F5E4F3EF
EEC0C0C0406B6F616C61407061644 04D407365
6C65637440C0EAEFF9F3F4E9,570
1030 DATA E3EBC0CDCDCDC0F3F4E1F2F4C000
002F320030322533330034322927272532 0000
707046452E4D00308D8D8D8D,161
1040 DATA 8D8D8D8D8D8D8D8D8D8D8D8D8D8D
8D8D8D8D8D8D8D8D8D8D8D8D8D8D8D8D8D8D8D
8D8D8D8D8D8D8D8D8D8D8D8D,506
1050 DATA 8D8D8D8D8D8D8D8D8D8D8D8D8D8D
8D8D8D8D8D8D8D8D8D8D8D8D8D8D8D8D8D8D8D
8D8D8D8D8D8D8D8D8D8D8D8D,851
1060 DATA 8D41A820D8A9008D0ED4A27F9580
CAD0FBE6BBA9FF859C85A985B985BA20EA298D
07D2A9C48DC402A9848DC502,315
1070 DATA A90A8DC602A9988DC702A9508D00
02A9238D0102A9C08D0ED4A9328D00D48D2F02
A9008D1DD08D08D2A2049D0D,970
1080 DATA D0CA10FA2065E4A9008D3002A920
8D3102A223A076A907205CE4A93C858CA58CD0
FCAD7C024D7D02D010AD8402,64
1090 DATA F00BAD1FD02903C903F0EA290185
B0A90A858CA58CD0FCAD1FD0C907D0F1A93085
81A900858 0A20FA89180C8D0,21
1100 DATA FBE681CA10F6A2009D00039D0004
9D00059D00069D0007CAD0EEA9A88D3002A920
8D3102A9008D07D4A93E8D2F,584
1110 DATA 028D00D4A9038D1DD0A9118D6F02
A90085BBA98F858285 84A937858385 85A200A0
00BD5F22D0034C9522300991,324
1120 DATA 829184C8E84C11220A85870A0AA9
55B002A90048BD5F22290F858668918 29184C8
C686D0F7E8A58710CB38A582,647
1130 DATA E9288582B002C68318A584692885
8490B5E6854C0F22EAEAEAEAEA15A854C115A8
54C105A850C105A850C101A8,397
1140 DATA 40C181E88115A654C18105A650C1
8101A640C182E68205A450C183E484E200A240
A00018A9607D0F2399401F9D,638
1150 DATA 801FA9507DCE229D801E99401E38
A950FDCE229D801E99C01EA960FD0F239D001F
99C01FC8CA10CE4C34250001,255
1160 DATA 02020304050506070809090A0B0C
0C0D0E0E0F101011121213141415151617171 8
18191 91A1A1B1B1B1C1C1D1D,15
1170 DATA 1D1E1E1E1E1F1F1F1F1F20202020
2020202036363636363636363535353434 3433
33323231313 02F2F2E2D2C2C,943
1180 DATA 2B2A2928272626252423 21201F1E
1D1C1B1A18171615141211100F0D0C0B090807
05040301004 88A48E6AAA5AA,834
1190 DATA 2907AABD6E2305B18D0AD48D16D0
A98C8D17D068AA684000020406080604 02D8A6
ABBD8F2C8DC302A90585AAA9,370
1200 DATA C08D0ED4ADFC02C921D00BA5B249
FF85B2A9FF8DFC02A5B2F00DA900A2079D00D2
CA10FA4C62E4A5BBD009ADC6,721
1210 DATA 0218691 08DC602A5B8F00C4A4A8D
07D2A9288D06D2C6B8A5A910034C3324A491A2
05A90F39FD0299FD02C8CA10,578
1220 DATA F4AD7802A690A4914AB00288884A
B002C8C84AB001CA4AB001E8E0309006E0D0B0
028690C0209006C0E0B00284,258
1230 DATA 91A5B0F003207B2BA5BCD01CA205
A491BD892C19FD0299FD02C8CA10F3A690CA8E
07D0E8E88E06D0A5AEF002C6,437
1240 DATA AEA58CF002C68CA58DF002C68DA5
8EF002C68EA5A910034C62E4A5A5F03AE689A4
8918B9001E8596692F8D05D0,351
1250 DATA 69028D04D0B9001F4A85972A6924
AAE68FA58F2908A8A9088588BD000329F01924
259D0003CAC8C688D0EFA5AB,332
1260 DATA F030ACF42D8888A209C020900AC0
DFB006BD912C99000788CA10EEADE82D8D03D0
E6ACA5AC4A2903AABD9B2C8D,987
1270 DATA 952C8D962CA6BA1007A9008D01D2
F00DA9A68D01D2BD752D8D00D2C6BAA6B91007
A9008D03D2F00DA9A68D03D2,276
1280 DATA BD832D8D02D2C6B9A5ABF00BADF4
2DC9249004C9E79007A9008D05D2F017E6B7A6
B7E00C3004A20086B7A9A88D,189
1290 DATA 05D2BD912D8D04D24C62E4000000
0A040A000000000000040A040000A900859D859E
859F858C85B285BD85AB859B,122
1300 DATA A20B9DD92DCA10FAA213BD0D2D9D
452ECA10F7A901859A85A5A904859CA90C8DC6
02A9348DC0028DC1028DC202,315
1310 DATA A97F8590A9818591A90185A92048
2BA9548D9337A9158D94378D1ED0205E2BA69B
BD212D85C6BD2D2D85C7BD39,537
1320 DATA 2D85ADBD452DC9FFD005AD0AD229
F085B1BD5D2D85A6BD512D85A7BD692D85A8E0
0BF002E69BF8A59A18690185,417
1330 DATA 9AD8A5B2D0FCA900854DA5A91007
A5BDD0034CD229AD93372903C903F009AD9437
29C0C9C0D00FA90085C685A5,113
```

```
1340 DATA A9FF859C85A920EA29AD1FD0C907
F0034C1221204327A5A5F010AD8402C599F009
8599C900D003206528 20AE28,172
1350 DATA A58DD01720802920F62920762620
1B29A5ABF003202328A901858DA5C6D089A5A9
3085ADD92D0DDA2D0DDB2D0D,871
1360 DATA DC2DF0034CCF254C8F25A94085B8
E6BDA4BDA5BE999D2DA5BF99B12DA90099C52D
60A90085C1E6C1A5BDC5C130,88
1370 DATA F3A6C1A90085C0BDC52DC925302E
E6C038E925C92530258AA8E8E4BDF0021015BD
9D2D999D2DBDB12D99B12DBD,973
1380 DATA C52D99C52DC8D0E4C6BDC6C14C7A
26FEC52DA8BD9D2D1879C32C85C2C9A0B0ACBD
B12D79E82C85C3C960B0A020,536
1390 DATA F326A5C0D00ABDAF2C118A918A4C
7A26BDB32C318A4CE626A5C30A858AA900858B
068A268B068AA58A85C5268B,774
1400 DATA A58B85C4068A268B068A268BA58A
1865C5858AA58B65C4858BA90018658A858AA9
30658B858BA5C22903AAA5C2,182
1410 DATA 4A4A18658A858A9002E68BA00060
A5AED004A5C6D00160A203BDD92DF004CA10F8
60A9019DD92DC6C6A9009D15,608
1420 DATA 2E9D212EA5A93053E003D04FAD0A
D2C5ADB048A90185ABAD0AD22903A8B99F2CC9
FFD0052019286923859 29DE5,918
1430 DATA 2DB9A32CC9FFD00520192869 3785
939DF12DB9A72CC9FFD005A9E638E5938594B9
AB2CC9FFD002A59285954C01,917
1440 DATA 28AD0AD23019AD0AD22901A8B967
2C9DF12DAD0AD2C9FAB0F99DE52D4CF127AD0A
D22901A8B9672C9DE52DAD0A,406
1450 DATA D2C9FAB0F99DF12DBDE52D8592BD
F12D8593A98085948595 20EE2AA5CA9D2D2EA5
CB9D392EA5C89DFD2DA5C99D,190
1460 DATA 092E60AD0AD2297FC964B0F760AD
0AD2C906B00DA207BDDD2DF007CABDDD2DF001
60A93085 95A95085 94ADE82D,855
1470 DATA 38E92C85929DE92DC9A0B0E8ADF4
2DE9254A85939DF52DC95FB0D9A90D85B94C91
28A205BDDD2DF004CA10F860,82
1480 DATA A90D85BAA59685929DE92DA59785
939DF52DA59038E9308594A59138E9204A8595
20EE2AA5C89D012EA5C99D0D,986
1490 DATA 2EA5CA9D312EA5CB9D3D2EA9019D
DD2D60A58ED0FBA59C1004A901D002A5C7858E
A203BDD92DF052208F2AA59C,574
1500 DATA 1009208F2A208F2A208F2ABDF12D
1869028582A900858A8A0904858B8684E003D0
04A5ABD027BC2D2EA9118587,3
1510 DATA BE852CA482C0209009C0DFB005BD
692C918A88CAC687D0EDA684BDE52D9D00D0CA
10A660A20BBDD92DF058BDE5,350
1520 DATA 2D85C2BDF12D85C3869820F326B1
8A3DB32C918AA698208F2ABDE52DC9A0B02285
C2BDF12DC960B01985C320F3,478
1530 DATA 26BDAF2C318AF015A698A5C285BE
A5C385BF205E26A9009DD92D4C7A29BDAF2CA6
983DB72C118A918ACAE003D0,136
1540 DATA 9E60A58CF00160A59C30FBA90185
A5AD08D00D09D0F0EFA90085A58589A69C9D53
2EC69C1009A9FF85C685A920,31
1550 DATA EA29A59685BEA59785BF205E26A9
508596A9158597A200BD000329F09D0003CAD0
F5A9FF858C6020EA29AD8402,611
1560 DATA 2D7C022D7D02F007AD1FD0C907F0
EE4C1221A9008D01D28D03D28D05D260A203A5
ABF01EA90085AFA5A9304DAD,633
1570 DATA E82DC9279046C9D3B042ADF42DC9
13903BC9E7B037A90085AFBD04D02905F063E6
AF2904F026A5A93022A90285,277
1580 DATA AEE003D00BA5ABF007A5A885A14C
4A2AA5A685A2A5A785A1869820362BA698A900
9DD92DBC2D2EBDE52D38F987,54
1590 DATA 2C85BEBDF12D38E9284A85BFA5AB
F00FE003D00BA90085AB20DA2AA5A9300A20DA
2AA5AFF003205E26CA108F8D,358
1600 DATA 1ED060BD152E187DFD2D9D152EA9
002A85CCBD212E7D092E9D212EA9002A85CDBD
2D2EF008BDE52D65CC4CBF2A,5
1610 DATA BDE52D38E5CC9DE52DBD392EF009
BDF12D38E5CD4CD62ABDF12D1865CD9DF12D60
A9009D00D0A88A0904858B98,847
1620 DATA 858A918A88D0FB60A90085CAA592
C5949005E5944C042BE6CAA59438E59285C8A9
0185CBA593C5959005E5954C,177
1630 DATA 1C2BC6CBA59538E59385C905C8D0
04A98085C8A5C8300BA5C9300706C806C94C26
2B60A000F818A202B59D75A0,498
1640 DATA 959D94A0CA10F5D8A91085A4A201
A000B99D0020662BE8E8C8C003D0F360A05084
A4A59AA20B85A3290F05A49D,227
1650 DATA 462EA5A34A4A4A4A05A49D452E60
A204B5CD95CEB5D295D3CAD0F5A491A205B900
0329F0990003CA10F5A90185,684
1660 DATA BCAD700285CEC9059052AD710285
D3C9059049A9CE85B5A90085B620F72BB03C69
10C930B002A930C9D09002A9,205
1670 DATA CF8590A9D385B5A90085B620F72B
B01F6910C920B002A920C9E09002A9DF8591AD
7C024D7D0249018D8402A900,391
1680 DATA 85BC60A900A20495DDCA10FB85B3
85B4A8A201B1B599D800C8205A2CB1B50A36DD
71B59002F6DD0A36DD95D8E8,463
1690 DATA C8205A2CB1B599D800A204B5D865
B385B3B5DD65B485B4CA10F1A204A5B346B46A
CAD0FAAAA00438F1B5B00449,757
1700 DATA FF6901C918B0048A8810EF60B1B5
0A36DD0A36DD95D8E8C86000FA000000000000
DC3E7E3EDC00000000000076,27
1710 DATA F8FCF876000000000000001B102F2A
4040A0A0404034F800001 87E000007E18000092
49249228FFD2FFFF14FFE6FF,377
1720 DATA D2FF2814FFE6FFC0300C033FCFF3
FC00000000FFFFFFFFFFFFAAAA0001FF00FF01
0002FFFE000100FE020101FF,69
1730 DATA 0002FEFF0300FDFE0302FFFE01FF
03FD01FD02000001FF00010100FF0102FFFEFF
0102FE0203FF00FE01FD00FE,931
1740 DATA FF020302FDFD000103FFFE000000
0000000000006C766C000000CACACACACA000A0F
141914190F141914191E0C0B,829
1750 DATA 0A09080707060505040300 0A325A
3250283C6450787D20304050607080A0B0C0D0
FF000000000000000000000101,251
1760 DATA 0110203040506070809000102000
0101010202030303040404F4FED2DCB0BA8E98
6C764A54283265605550 4540,601
1770 DATA 353025201510050000A0B0C0E1011
1211100E0C0B000000000000000000000000000
000000000000000000000000,981
```

CHECKSUM DATA
(see p. 15)

```
10 DATA 66,351,496,811,423,729,200,603
,555,573,694,613,29,205,957,7305
160 DATA 748,198,962,626,491,30,155,11
4,408,59,873,219,658,823,822,7186
1060 DATA 217,905,57,111,816,713,717,9
24,566,982,425,338,370,877,257,8275
1210 DATA 986,698,117,994,961,216,869,
75,578,126,47,200,842,795,861,8365
1360 DATA 413,206,503,133,992,885,976,
63,302,513,171,449,151,136,685,6578
1510 DATA 215,972,203,79,973,897,29,35
0,93,25,458,913,865,962,895,7929
1660 DATA 841,809,905,932,939,702,412,
8,684,35,192,678,7137
```

Assembly language listing

```
;
; File: PLANET1.M65
;
        .OPT NO LIST
; -------------------
; ANALOG Computing's
; PLANETARY  DEFENSE
; -------------------
;
; by Charles Bachand
;   and Tom Hudson
;
; Written with OSS MAC/65
;
; -------------------
; Hardware Registers
; -------------------
;
```

Great Software • Great Service • ~~Great~~ incredible Prices

from Atari

Pole Position

ROM Cart. $36.88

Not since the release of their Pacman cartridge has there been such great anticipation for a game program from Atari. Expectations increased though months of pre-advertising without product delivery. Well, Pole Position is out now and it's well worth the wait. It's an exciting rendition of the coin-op game that was one of the two most popular (the other is Q*Bert) of 1983.
All Atari programs including Ms. Pacman, Dig Dug, Qix and Donkey Kong in stock and available at similar savings.

from Electronic Arts

Pinball Construction Set

48K Disk $27.88

Steve Wozniak of Apple Computer Corp. has called Bill Budge's Pinball Construction Set "the best program ever written for an 8-bit computer." You can build your own pinball machine from the parts provided — bumpers, flippers, targets, etc. — and, then, save and play them. You can also control the machine's logic, affecting the scoring, the sound, or even the amount of gravity inside the machine!
All Electronic Arts programs including Archon, M.U.L.E. and Murder on the Zinderneuf in stock and available at similar savings.

from Parker Brothers

Q*Bert

ROM Cart. $33.88

It's everybody's favorite arcade character of 1983. It's lovable Q*Bert. This is the official Atari cartridge version of the smash coin-op hit. Excellent graphics and game play.
All Parker Brothers programs including Popeye, Super Cobra, Frogger and Astro Chase in stock and available at similar savings.

from Broderbund

Lode Runner

48K Disk $22.88

Runner-up (to Miner 2049er) as the outstanding original arcade game of 1983 in our in-house poll. 150 thrill-packed levels plus a game generator feature which permits you to design and play your own levels. If you're starting to "burn out" on arcade games, this program will rekindle your enthusiasm.
All Broderbund Software including The Arcade Machine, Operation Whirlwind and David's Midnight Magic in stock and available at similar savings.

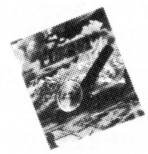
from Synapse

Blue Max

32K Tape or Disk $22.88

The premier Atari software house brings you still another fabulous program. As a member of the Royal Air Force, you will command a biplane fighter/bomber and will battle the Axis on the ground and in the skies. Excellent scrolling and 3-D graphics.
All Synapse programs including Rainbow Walker, Encounter and Pharoah's Curse in stock and available at similar savings.

from Microprose



Solo Flight

48K Tape or Disk $22.88

The first real flight simulator for the Atari. Features include: color, three-dimensional graphics; dual radio navigation instruments; full-feature, actual configuration instrument panel; multiple air navigation charts; multiple airports and runways; day, night, cross-wind and instrument flying scenarios; takeoffs and landings; emergency procedure practice; and (as they say) much, much more. An outstanding program that realistically simulates the thrill and challenge of flying.
All Microprose Software including NATO Commander, Hellcat Ace and Floyd of the Jungle in stock and available at similar savings.

NEW RELEASES
Save Big on a Dozen Recent Releases!

Bank St. Writer (Broderbund) D	$44.88
Eagles (SSI) D	$27.88
Enchanter (Infocom) D	$34.88
Infidel (Infidel) D	$34.88
LOGO (Atari) R	$74.88
Oil's Well (Sierra On-Line) R	$23.88
Pitstop (Epyx) R	$27.88
Planetfall (Infocom) D	$34.88
Spellunker (Micrographic) D	$27.88
Trains (Spinnaker) D	$27.88
Ultima III (Origin) D	$39.88
Zaxxon (Datasoft) T&D	$24.88

SUPERSPECIALS
Save 33% to 43% on these Classic Programs!

Baja Buggies (Gamestar) T&D	$19.88
Bandits (Sirius) D	$19.88
Castle Wolfenstein (Muse) D	$18.88
Choplifter (Broderbund) D	$22.88
Crush, Crumble & Chomp (Epyx) T&D	$18.88
Facemaker (Spinnaker) D	$22.88
Jumbo Jet Pilot (Thorn-EMI) R	$29.88
Labyrinth (Broderbund) T&D	$18.88
Mastertype (Lightning) D	$24.88
Serpentine (Broderbund) T&D	$19.88
Soccer (Thorn-EMI) R	$29.88
Visicalc (Visicorp) D	$139.88

Call or write for FREE CATALOG (specify Atari)

Mail order prices do not apply at our retail location.

HOW TO ORDER:

Use mail or phone. For immediate shipment pay by Money Order, Certified Check, Cashiers Check, VISA, or Master Card (add 4% for credit card). Personal checks are accepted but require 1-3 weeks to clear. 10% deposit required for C.O.D.

SHIPPING & HANDLING:

$2.50 for orders under $50; free shipping over $50. Mass. residents add 5% sales tax. Prices are mail order only and are subject to change without notice.

SOFTWARE PLUS

(617) 369-1992

for orders and information
phone hours: 10-6 EST Mon.—Sat.

45 Walden St., Box AA, Concord, Mass. 01742

```
HPOSPØ  =  $DØØØ   ;PØ horizontal
HPOSMØ  =  $DØØ4   ;MØ horizontal
PØPF    =  $DØØ4   ;PØ-PField coll.
MØPL    =  $DØØ8   ;MØ-PLayer coll.
GRAFPØ  =  $DØØD   ;PM graphics
COLPFØ  =  $DØ16   ;PF color Ø
GRACTL  =  $DØ1D   ;graphic control
HITCLR  =  $DØ1E   ;clear hit reg
CONSOL  =  $DØ1F   ;console buttons
AUDF1   =  $D2ØØ   ;frequency 1
AUDC1   =  $D2Ø1   ;volume 1
AUDCTL  =  $D2Ø8   ;audio control
RANDOM  =  $D2ØA   ;random numbers
DMACTL  =  $D4ØØ   ;DMA control
PMBASE  =  $D4Ø7   ;PM base address
WSYNC   =  $D4ØA   ;wait hor. sync
NMIEN   =  $D4ØE   ;interrupt reg.
ATRACT  =  $4D     ;attract flag
;
; -------------------------
; Operating System Vectors
; -------------------------
;
SETVBV  =  $E45C   ;set v.blank
XITVBV  =  $E462   ;exit v.blank
SIOINV  =  $E465   ;serial I/O init
;
; ----------------
; Shadow Registers
; ----------------
;
VDSLST  =  $Ø2ØØ   ;DLI vector
SDMCTL  =  $Ø22F   ;DMA control
SDLSTL  =  $Ø23Ø   ;DList pointer
GPRIOR  =  $Ø26F   ;gr. priority
PADDLØ  =  $Ø27Ø   ;paddle Ø
PADDL1  =  $Ø271   ;paddle 1
STICKØ  =  $Ø278   ;joystick Ø
PTRIGØ  =  $Ø27C   ;paddle trig Ø
PTRIG1  =  $Ø27D   ;paddle trig 1
STRIGØ  =  $Ø284   ;stick trig Ø
PCOLRØ  =  $Ø2CØ   ;player colors
COLORØ  =  $Ø2C4   ;playfield "
CH      =  $Ø2FC   ;keyboard char
;
; -------------------
; Page Zero Registers
; -------------------
;
        *=  $8Ø    ;top of page Ø
;
INDEX   *=  *+2    ;temp index
INDX1   *=  *+2    ;temp index
INDX2   *=  *+2    ;temp index
COUNT   *=  *+1    ;temp register
TEMP    *=  *+1    ;temp register
SATEMP  *=  *+1    ;temp register
SCNT    *=  *+1    ;orbit index
LO      *=  *+1    ;plot low byte
HI      *=  *+1    ;plot high byte
DEADTM  *=  *+1    ;death timer
EXPTIM  *=  *+1    ;explosion timer
BOMTIM  *=  *+1    ;bomb timer
SATPIX  *=  *+1    ;sat. pic cntr
CURX    *=  *+1    ;cursor x
CURY    *=  *+1    ;cursor y
FROMX   *=  *+1    ;vector from X
FROMY   *=  *+1    ;vector from Y
TOX     *=  *+1    ;vector to X
TOY     *=  *+1    ;vector to Y
SATX    *=  *+1    ;satellite x
SATY    *=  *+1    ;satellite y
XHOLD   *=  *+1    ;x reg hold area
LASTRG  *=  *+1    ;last trigger
LEVEL   *=  *+1    ;level number
BLEVEL  *=  *+1    ;binary level #
LIVES   *=  *+1    ;lives left
SCORE   *=  *+3    ;score digits
SCOADD  *=  *+3    ;score inc.
SHOBYT  *=  *+1    ;digit hold
SHCOLR  *=  *+1    ;digit color
SATLIV  *=  *+1    ;satellite flag
BOMVL   *=  *+1    ;bomb value low
BOMVH   *=  *+1    ;bomb value high
SAUVAL  *=  *+1    ;saucer value
GAMCTL  *=  *+1    ;game ctrl flag
DLICNT  *=  *+1    ;DLI counter
SAUCER  *=  *+1    ;saucer flag
SAUTIM  *=  *+1    ;image timer
SAUCHN  *=  *+1    ;saucer chance
BOMBWT  *=  *+1    ;bomb wait time
BOMCOL  *=  *+1    ;bomb collis flg
DEVICE  *=  *+1    ;koala pad sw
PLNCOL  *=  *+1    ;planet color
PAUSED  *=  *+1    ;pause flag
AVG     *=  *+2    ;average
PTR     *=  *+2    ;queue pointer
SSSCNT  *=  *+1    ;saucer snd cnt
EXSCNT  *=  *+1    ;expl. snd count
ESSCNT  *=  *+1    ;enemy shot snd
PSSCNT  *=  *+1    ;player shot snd
TITLE   *=  *+1    ;title scrn flag
PENFLG  *=  *+1    ;pen up/dwn flg
EXPCNT  *=  *+1    ;explosion counter
NEWX    *=  *+1    ;explosion x
NEWY    *=  *+1    ;explosion y
PLOTCLR *=  *+1    ;plot/erase flag
COUNTR  *=  *+1    ;explosion index
PLOTX   *=  *+1    ;plot x coord
PLOTY   *=  *+1    ;plot y coord
HIHLD   *=  *+1    ;plot work area
LOHLD   *=  *+1    ;plot work area
BOMBS   *=  *+1    ;bombs to come
BOMTI   *=  *+1    ;bomb speeds
VXINC   *=  *+1    ;vector x hold
VYINC   *=  *+1    ;vector y hold
LR      *=  *+1    ;vector left/right hold
UD      *=  *+1    ;vector up/down hold
DELTAX  *=  *+1    ;vector work area
DELTAY  *=  *+1    ;vector work area
XQ      *=  *+5    ;x queue
YQ      *=  *+5    ;y queue
SL      *=  *+5    ;samples lo
SH      *=  *+5    ;samples hi
;
; ---------------------------
; Screen + Player/Missile Area
; ---------------------------
;
SCRN    =  $3ØØØ   ;screen area
PPOS    =  SCRN+1935 ;planet pos
;
PM      =  $ØØ
MISL    =  PM+$Ø3ØØ ;missiles
PLRØ    =  MISL+$Ø1ØØ ;player Ø
PLR1    =  PLRØ+$Ø1ØØ ;player 1
PLR2    =  PLR1+$Ø1ØØ ;player 2
PLR3    =  PLR2+$Ø1ØØ ;player 3
;
ORBX    =  $1EØØ   ;orbit X
ORBY    =  $1FØØ   ;orbit Y
;
; -------------
; Start of game
; -------------
;
        *=  $2ØØØ
;
; ------------------
; Intro Display List
; ------------------
;
TLDL    .BYTE $7Ø,$7Ø,$7Ø,$7Ø
        .BYTE $7Ø,$7Ø,$7Ø,$7Ø
        .BYTE $7Ø,$46
        .WORD MAGMSG
        .BYTE $7Ø,7,$7Ø,6
        .BYTE $1Ø,6,$7Ø,$7Ø
        .BYTE 6,$2Ø,6,$4Ø
        .BYTE 6,$41
        .WORD TLDL
;
; ------------------
; Intro Message Text
; ------------------
;
MAGMSG  .SBYTE " ANALOG CO"
        .SBYTE "MPUTING'S "
        .SBYTE +$4Ø," PLANETARY"
        .SBYTE +$4Ø,"  DEFENSE "
        .SBYTE +$8Ø," BY CHARLE"
        .SBYTE +$8Ø,"S BACHAND "
        .SBYTE +$CØ,"   AND TOM"
        .SBYTE +$CØ," HUDSON   "
        .SBYTE +$4Ø," KOALA PAD"
        .SBYTE +$4Ø," - SELECT "
        .SBYTE +$CØ," JOYSTICK "
        .SBYTE +$CØ,"--- START "
        .SBYTE "  OR PRESS"
        .SBYTE " TRIGGER  "
;
; -----------------
; Game Display List
; -----------------
;
GLIST   .BYTE $7Ø,$7Ø,$46
        .WORD SCOLIN
        .BYTE $4D
        .WORD SCRN
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$8D
        .BYTE $8D,$8D,$8D,$41
        .WORD GLIST
;
; ----------------------
; Display the Intro Screen
; ----------------------
;
PLANET  CLD              ;clear decimal
        LDA #$ØØ         ;get zero
        STA NMIEN        ;display off
        LDX #$7F         ;set index
CLPGØ   STA $8Ø,X        ;clr top page Ø
        DEX              ;dec pointer
        BNE CLPGØ        ;done? No.
        INC TITLE        ;title on flag
        LDA #$FF         ;get $FF
        STA LIVES        ;set dead
        STA GAMCTL       ;game control
        STA ESSCNT       ;no enemy shots
        STA PSSCNT       ;no player shots
        JSR SNDOFF       ;no sound on 123
        STA AUDC1+6      ;turn off snd 4
        LDA #$C4         ;medium green
        STA COLORØ       ;score color
        LDA #$84         ;medium blue
        STA COLORØ+1     ;text color
        LDA #$ØA         ;bright white
        STA COLORØ+2     ;shot color
        LDA #$98         ;light blue
        STA COLORØ+3     ;text color
        LDA # <DLI       ;set DLI vector
        STA VDSLST       ;low byte
        LDA # >DLI       ;set DLI vector
        STA VDSLST+1     ;high byte
        LDA #$CØ         ;enable Display
        STA NMIEN        ;List Interrupts
        LDA #$32         ;PM DMA off
        STA DMACTL       ;DMA control
        STA SDMCTL       ;and shadow reg
        LDA #Ø           ;get zero
        STA GRACTL       ;graphics ctrl
        STA AUDCTL       ;reset audio
        LDX #4           ;5 PM registers
NOPM    STA GRAFPØ,X     ;clr register
        DEX              ;dec index
        BPL NOPM         ;done? No.
        JSR SIOINV       ;init sound
        LDA # <TLDL      ;DL addr low
        STA SDLSTL       ;DL pntr low
        LDA # >TLDL      ;DL addr high
        STA SDLSTL+1     ;DL pntr high
        LDX # >VBLANK    ;vblank high
        LDY # <VBLANK    ;vblank low
        LDA #7           ;deferred
        JSR SETVBV       ;set vblank
        LDA #6Ø          ;one second
        STA DEADTM       ;dead time
;
; -------------------------
; Check console and triggers
; -------------------------
;
START   LDA DEADTM       ;look dead time
        BNE START        ;alive? No.
CCHK    LDA PTRIGØ       ;paddle trig Ø
        EOR PTRIG1       ;mask w/PTRIG1
        BNE PDEV         ;pushed? Yes.
        LDA STRIGØ       ;stick trig
        BEQ PDEV         ;pushed? Yes.
        LDA CONSOL       ;get console
        AND #3           ;do START/SELECT
        CMP #3           ;test buttons
        BEQ CCHK         ;any pushed? No.
        AND #1           ;mask off START
PDEV    STA DEVICE       ;device switch
RELWT   LDA #1Ø          ;1/6 second
        STA DEADTM       ;dead time
RELWT2  LDA DEADTM       ;debounce!
        BNE RELWT2       ;time up? No.
        LDA CONSOL       ;get console
        CMP #7           ;keys released?
        BNE RELWT        ;No. loop until
;
; --------------------------
; Clear PM Area and Playfield
; --------------------------
;
        LDA # >SCRN      ;scrn addr high
        STA INDEX+1      ;pointer high
        LDA #Ø           ;get zero
        STA INDEX        ;pointer low
        LDX #15          ;16 pages Ø..15
        TAY              ;use as index
CLØ     STA (INDEX),Y    ;clear ram
        INY              ;next byte
        BNE CLØ          ;page done? No.
        INC INDEX+1      ;next page
```

```
         DEX           ;page counter
         BPL CLØ       ;scrn done? No.
         LDX #Ø        ;now clear P/m
CLPM     STA MISL,X    ;clear missiles
         STA PLRØ,X    ;clear plyr Ø
         STA PLR1,X    ;clear plyr 1
         STA PLR2,X    ;clear plyr 2
         STA PLR3,X    ;clear plyr 3
         DEX           ;done 256 bytes?
         BNE CLPM      ;no, loop back!
         LDA # <BLIST  ;Point to the
         STA SDLSTL    ;game display
         LDA # >BLIST  ;list to show
         STA SDLSTL+1  ;the playfield.
         LDA # >PM     ;PM address high
         STA PMBASE    ;into hardware
         LDA #$3E      ;enable single
         STA SDMCTL    ;line resolution
         STA DMACTL    ;DMA control
         LDA #3        ;enable player
         STA GRACTL    ;and missile DMA
         LDA #$11      ;set up
         STA GPRIOR    ;P/M priority
         LDA #Ø        ;get zero
         STA TITLE     ;title off
;
; ---------------
; Draw The Planet
; ---------------
;
         LDA # <PPOS   ;planet pos high
         STA INDX1     ;pointer #1 low
         STA INDX2     ;pointer #2 low
         LDA # >PPOS   ;planet pos high
         STA INDX1+1   ;pointer #1 high
         STA INDX2+1   ;pointer #2 high
         LDX #Ø        ;table pointer
DPØ      LDY #Ø        ;index pointer
DP1      LDA DPTBL,X   ;table value
         BNE DP2       ;done? No.
         JMP SETUP     ;continue
DP2      BMI DPRPT     ;repeat? Yes.
         STA (INDX1),Y ;put values
         STA (INDX2),Y ;onto screen
         INY           ;inc index pntr
         INX           ;inc table pntr
         JMP DP1       ;continue
;
; -------------------
; Repeat Byte Handler
; -------------------
;
DPRPT    ASL A         ;shift byte
         STA TEMP      ;new line flag
         ASL A         ;NL bit -> carry
         ASL A         ;color -> carry
         LDA #$55      ;color 1 bits
         BCS FILL1     ;color 1? Yes.
         LDA #Ø        ;get background
FILL1    PHA           ;save color byte
         LDA DPTBL,X   ;table value
         AND #$ØF      ;mask 4 bits
         STA COUNT     ;save as count
         PLA           ;restore color
FILL2    STA (INDX1),Y ;put bytes
         STA (INDX2),Y ;onto screen
         INY           ;inc index
         DEC COUNT     ;dec byte count
         BNE FILL2     ;done? No.
         INX           ;inc table index
         LDA TEMP      ;get flag
         BPL DP1       ;new line? No.
         SEC           ;set carry
         LDA INDX1     ;Yes. get low
         SBC #4Ø       ;subtract 4Ø
         STA INDX1     ;new low
         BCS DPN1      ;overflow? No.
         DEC INDX1+1   ;decrement high
DPN1     CLC           ;clear carry
         LDA INDX2     ;get low
         ADC #4Ø       ;add 4Ø
         STA INDX2     ;new low
         BCC DPØ       ;overflow? No.
         INC INDX2+1   ;increment high
         JMP DPØ       ;continue
;
; ----------------
; Planet Draw Data
; ----------------
;
DPTBL    .BYTE $EA,$EA,$EA,$EA
         .BYTE $EA,$15,$A8,$54
         .BYTE $C1,$15,$A8,$54
         .BYTE $C1,$Ø5,$A8,$5Ø
         .BYTE $C1,$Ø5,$A8,$5Ø
         .BYTE $C1,$Ø1,$A8,$4Ø
         .BYTE $C1,$81,$E8,$81
         .BYTE $15,$A6,$54,$C1
         .BYTE $81,$Ø5,$A6,$5Ø
         .BYTE $C1,$81,$Ø1,$A6
         .BYTE $4Ø,$C1,$82,$E6
         .BYTE $82,$Ø5,$A4,$5Ø
         .BYTE $C1,$83,$E4,$84
         .BYTE $E2,Ø
;
; --------------------------
; Setup Orbiter Coordinates
; --------------------------
;
SETUP    LDX #64       ;do 65 bytes
         LDY #Ø        ;quad 2/4 offset
SU1      CLC           ;clear carry
         LDA #96       ;center Y
         ADC OYTBL,X   ;add offset Y
         STA ORBY+$4Ø,Y ;quad-2 Y
         STA ORBY+$8Ø,X ;quad-3 Y
         LDA #8Ø       ;center X
         ADC OXTBL,X   ;add offset X
         STA ORBX,X    ;quad-1 X
         STA ORBX+$4Ø,Y ;quad-2 X
         SEC           ;set carry
         LDA #8Ø       ;center X
         SBC OXTBL,X   ;sub offset X
         STA ORBX+$8Ø,X ;quad-3 X
         STA ORBX+$CØ,Y ;quad-4 X
         LDA #96       ;center Y
         SBC OYTBL,X   ;sub offset Y
         STA ORBY,X    ;quad-1 Y
         STA ORBY+$CØ,Y ;quad-4 Y
         INY           ;quad 2/4 offset
         DEX           ;quad 1/3 offset
         BPL SU1       ;done? No.
         JMP INIT      ;continue
;
; ----------------------------
; Orbiter X,Y Coordinate Data
; ----------------------------
;
OXTBL    .BYTE Ø,1,2,2,3
         .BYTE 4,5,5,6,7
         .BYTE 8,9,9,1Ø,11
         .BYTE 12,12,13,14,14
         .BYTE 15,16,16,17,18
         .BYTE 18,19,2Ø,2Ø,21
         .BYTE 21,22,23,23,24
         .BYTE 24,25,25,26,26
         .BYTE 27,27,27,28,28
         .BYTE 29,29,29,3Ø,3Ø
         .BYTE 3Ø,3Ø,31,31,31
         .BYTE 31,31,32,32,32
         .BYTE 32,32,32,32,32
;
OYTBL    .BYTE 54,54,54,54,54
         .BYTE 54,54,54,53,53
         .BYTE 53,52,52,52,51
         .BYTE 51,5Ø,5Ø,49,49
         .BYTE 48,47,47,46,45
         .BYTE 44,44,43,42,41
         .BYTE 4Ø,39,38,38,37
         .BYTE 36,35,33,32,31
         .BYTE 3Ø,29,28,27,26
         .BYTE 24,23,22,21,2Ø
         .BYTE 18,17,16,15,13
         .BYTE 12,11,9,8,7
         .BYTE 5,4,3,1,Ø
;
; ----------------------
; Display list interrupt
; ----------------------
;
DLI      PHA           ;save Acc
         TXA           ;X --> Acc
         PHA           ;save X register
         INC DLICNT    ;inc counter
         LDA DLICNT    ;get counter
         AND #$Ø7      ;only 3 bits
         TAX           ;use as index
         LDA DLIBRT,X  ;planet bright
         ORA PLNCOL    ;planet color
         STA WSYNC     ;start of scan
         STA COLPFØ    ;color planet
         LDA #$8C      ;bright blue
         STA COLPFØ+1  ;shot color
         PLA           ;restore X
         TAX           ;Acc --> X
         PLA           ;restore Acc
         RTI           ;return
;
DLIBRT   .BYTE Ø,2,4,6 ;planet
         .BYTE 8,6,4,2 ;brightness
;
; -----------------------
; Vertical blank routine
; -----------------------
;
VBLANK   CLD           ;clear decimal
         LDX SAUCER    ;saucer flag as
         LDA P3COLR,X  ;index Ø or 1
         STA PCOLRØ+3  ;saucer color
         LDA #5        ;get 5
         STA DLICNT    ;reset DLI count
         LDA #$CØ      ;enable
         STA NMIEN     ;DLI's
         LDA CH        ;keyboard char
         CMP #$21      ;space bar?
         BNE PCHK      ;No. skip it
         LDA PAUSED    ;pause flag
         EOR #$FF      ;invert it
         STA PAUSED    ;save pause flag
         LDA #$FF      ;get $FF
         STA CH        ;reset keyboard
PCHK     LDA PAUSED    ;pause flag
         BEQ NOPAU     ;paused? No.
         LDA #Ø        ;get zero
         LDX #7        ;do 8 bytes
NOSND    STA AUDF1,X   zero sound
         DEX           ;dec index
         BPL NOSND     ;done? No.
         JMP XITVBV    ;exit VBLANK
NOPAU    LDA TITLE     ;title flag
         BNE NOCYC     ;title? Yes.
         LDA COLORØ+2  No. get color
         CLC           ;clear carry
         ADC #$1Ø      ;next color
         STA COLORØ+2  ;explosion col.
NOCYC    LDA EXSCNT    ;explosion cnt
         BEQ NOPAU2    ;any? No.
         LSR A         ;count/2
         LSR A         ;count/4
         STA AUDC1+6   ;explo volume
         LDA #4Ø       ;explosion
         STA AUDF1+6   ;explo frequency
         DEC EXSCNT    ;dec count
NOPAU2   LDA GAMCTL    ;game control
         BPL CURSOR    ;cursor? Yes.
         JMP TIMERS    ;No. skip
;
; --------------
; Cursor handler
; --------------
;
CURSOR   LDY CURY      ;get y pos
         LDX #5        ;clear 6 bytes
ERACUR   LDA #$ØF      ;now clear out
         AND MISL-3,Y  ;old cursor
         STA MISL-3,Y  ;graphics,
         INY           ;next Y position
         DEX           ;dec count
         BPL ERACUR    ;loop until done
         LDA STICKØ    ;read joystick
         LDX CURX      ;get X value
         LDY CURY      ;get Y value
         LSR A         ;shift right
         BCS NOTN      ;North? No.
         DEY           ;move cursor up
         DEY           ;two scan lines
NOTN     LSR A         ;shift right
         BCS NOTS      ;South? No.
         INY           ;cursor down
         INY           ;two scan lines
NOTS     LSR A         ;shift right
         BCS NOTW      ;West? No.
         DEX           ;cursor left
NOTW     LSR A         ;shift right
         BCS NOTE      ;East? No.
         INX           ;cursor right
NOTE     CPX #48       ;too far left?
         BCC BADX      ;Yes. skip next
         CPX #2Ø8      ;too far right?
         BCS BADX      ;Yes. skip next
         STX CURX      ;No. it's ok!
BADX     CPY #32       ;too far up?
         BCC BADY      ;Yes. skip next
         CPY #224      ;too far down?
         BCS BADY      ;Yes. skip next
         STY CURY      ;No. it's ok!
BADY     LDA DEVICE    ;KOALA switch
         BEQ NKOALA    ;KOALA PAD?
         JSR KOALA     ;Yes. do it
NKOALA   LDA PENFLG    ;koala pen flg
         BNE TIMERS    ;pen up? Yes.
         LDX #5        ;6 bytes...
         LDY CURY      ;get cursor Y
SHOCUR   LDA CURPIC,X  ;cursor pic
         ORA MISL-3,Y  ;mask missiles
         STA MISL-3,Y  ;store missiles
         INY           ;next scan line
         DEX           ;dec count
         BPL SHOCUR    ;done? No.
         LDX CURX      ;get x position,
         DEX           ;1 less for...
         STX HPOSMØ+3  ;missile 3
         INX           ;2 more for...
         INX           ;missile 2
         STX HPOSMØ+2  ;save position
;
; -------------------------
; Handle timers and orbit
; -------------------------
;
TIMERS   LDA BOMBWT    ;bomb wait cnt
         BEQ NOBWT     ;wait over? Yes.
         DEC BOMBWT    ;dec count
NOBWT    LDA DEADTM    ;death timer
         BEQ NOTIMØ    ;zero? yes.
         DEC DEADTM    ;decrement it!
NOTIMØ   LDA EXPTIM    ;exp timer
         BEQ NOTIM1    ;zero? Yes.
         DEC EXPTIM    ;decrement it!
NOTIM1   LDA BOMTIM    ;get bomb time
         BEQ NOTIM2    ;zero? Yes.
         DEC BOMTIM    ;dec bomb time
```

```
NOTIM2  LDA GAMCTL     ;game control
        BPL NOTOVR     ;game over? No.
        JMP XITVBV     ;exit VBLANK
NOTOVR  LDA SATLIV     ;get satellite
        BEQ NOSAT      ;alive? No.
        INC SCNT       ;inc count
        LDY SCNT       ;orbit index
        CLC            ;clear carry
        LDA ORBX,Y     ;get X coord
        STA SATX       ;save Pfield x
        ADC #47        ;X offset
        STA HPOSM0+1   ;horizontal pos
        ADC #2         ;+2 offset for
        STA HPOSM0     ;right side
        LDA ORBY,Y     ;get Y coord
        LSR A          ;divide by 2
        STA SATY       ;for playfield
        ROL A          ;restore for PM
        ADC #36        ;screen offset
        TAX            ;use as index
        INC SATPIX     ;next sat. image
        LDA SATPIX     ;get number
        AND #$08       ;use bit 3
        TAY            ;use as index
        LDA #8         ;do 8 bytes
        STA SATEMP     ;save count
SSAT    LDA MISL,X     ;missile graphic
        AND #$F0       ;mask off 1,2
        ORA SATSH,Y    ;add sat shape
        STA MISL,X     ;put player #1
        DEX            ;dec position
        INY            ;dec index
        DEC SATEMP     ;dec count
        BNE SSAT       ;done? No.
NOSAT   LDA SAUCER     ;saucer flag
        BEQ SOUNDS     ;saucer? No.
        LDY BOMBY+3    ;saucer Y pos
        DEY            ;-1
        DEY            ;-2
        LDX #9         ;10 scan lines
SSAULP  CPY #32        ;above top?
        BCC NXTSP      ;Yes. skip it
        CPY #223       ;below bottom?
        BCS NXTSP      ;Yes. skip it
        LDA SAUPIC,X   ;saucer image
        STA PLR3,Y     ;store player 3
NXTSP   DEY            ;next scan line
        DEX            ;dec index
        BPL SSAULP     ;done? No.
        LDA BOMBX+3    ;saucer X pos
        STA HPOSP0+3   ;move it
        INC SAUTIM     ;saucer time
        LDA SAUTIM     ;get counter
        LSR A          ;/2
        AND #$03       ;use only 0..3
        TAX            ;as X index
        LDA SAUMID,X   ;saucer middle
        STA SAUPIC+4   ;put in
        STA SAUPIC+5   ;saucer image
SOUNDS  LDX PSSCNT     ;shot sound
        BPL DOS1       ;shot? Yes.
        LDA #0         ;No. get zero
        STA AUDC1      ;volume for shot
        BEQ TRYS2      ;skip next
DOS1    LDA #$A6       ;shot sound vol
        STA AUDC1      ;set hardware
        LDA PLSHOT,X   ;shot sound
        STA AUDF1      ;frequency
        DEC PSSCNT     ;dec shot snd
TRYS2   LDX ESSCNT     ;enemy shots
        BPL DOS2       ;shots? Yes.
        LDA #0         ;No. get zero
        STA AUDC1+2    ;into volume
        BEQ TRYS3      ;skip rest
DOS2    LDA #$A6       ;shot sound vol
        STA AUDC1+2    ;set hardware
        LDA ENSHOT,X   ;shot sound
        STA AUDF1+2    ;frequency
        DEC ESSCNT     ;dec shot snd
TRYS3   LDA SAUCER     ;saucer flag
        BEQ NOS3       ;saucer? No.
        LDA BOMBY+3    ;saucer Y pos
        CMP #36        ;above top?
        BCC NOS3       ;Yes. skip
        CMP #231       ;below bottom?
        BCC DOS3       ;No. make sound
NOS3    LDA #0         ;get zero
        STA AUDC1+4    ;no saucer snd
        BEQ VBDONE     ;skip next
DOS3    INC SSSCNT     ;inc saucer cnt
        LDX SSSCNT     ;saucer count
        CPX #12        ;at limit?
        BMI SETS3      ;No. skip next
        LDX #0         ;get zero
        STX SSSCNT     ;zero saucer cnt
SETS3   LDA #$A8       ;saucer volume
        STA AUDC1+4    ;set hardware
        LDA SAUSND,X   ;saucer sound
        STA AUDF1+4    ;set hardware
VBDONE  JMP XITVBV     ;continue
;
; ---------------------
; Satellite shape table
; ---------------------
;
SATSH   .BYTE 0,0,0,$0A
        .BYTE $04,$0A,0,0
        .BYTE 0,0,0,$04
        .BYTE $0A,$04,0,0
;
        .INCLUDE #D:PLANET2.M65
        .INCLUDE #D:PLANET3.M65
        .END
```

```
;
; File: D:PLANET2.M65
;
; ----------------
; Initialize Misc.
; ----------------
;
INIT    LDA #0         ;zero out..
        STA SCORE      ;score byte 0
        STA SCORE+1    ;score byte 1
        STA SCORE+2    ;score byte 2
        STA DEADTM     ;dead timer
        STA PAUSED     ;pause flag
        STA EXPCNT     ;expl. counter
        STA SAUCER     ;no saucer
        STA BLEVEL     ;bomb level
        LDX #11        ;no bombs!
CLRACT  STA BOMACT,X   ;deactivate
        DEX            ;next bomb
        BPL CLRACT     ;done? No.
        LDX #19        ;zero score line
INISLN  LDA SCOINI,X   ;get byte
        STA SCOLIN,X   ;put score line
        DEX            ;next byte
        BPL INISLN     ;done? No.
        LDA #$01       ;get one
        STA LEVEL      ;game level
        STA SATLIV     ;live satellite
        LDA #4         ;get 4
        STA LIVES      ;number of lives
        LDA #$0C       ;set explosion
        STA COLOR0+2   ;brightness
        LDA #$34       ;medium red
        STA PCOLR0     ;bomb 0 color
        STA PCOLR0+1   ;bomb 1 color
        STA PCOLR0+2   ;bomb 2 color
        LDA #127       ;center screen X
        STA CURX       ;cursor X pos
        LDA #129       ;center screen Y
        STA CURY       ;cursor Y pos
        LDA #1         ;get one
        STA GAMCTL     ;game control
        JSR SHOSCO     ;display score
        LDA #$54       ;graphic-LF of
        STA SCRN+1939  ;planet center
        LDA #$15       ;graphic-RT of
        STA SCRN+1940  ;planet center
        STA HITCLR     ;reset collision
;
; ----------------------
; Set up level variables
; ----------------------
;
SETLVL  JSR SHOLVL     ;show level
        LDX BLEVEL     ;bomb level
        LDA INIBOM,X   ;bombs / level
        STA BOMBS      ;bomb count
        LDA INIBS,X    ;bomb speed
        STA BOMTI      ;bomb timer
        LDA INISC,X    ;% chance of
        STA SAUCHN     ;saucer in level
        LDA INIPC,X    ;planet color
        CMP #$FF       ;level >14?
        BNE SAVEPC     ;No. skip next
        LDA RANDOM     ;random color
        AND #$F0       ;mask off lum.
SAVEPC  STA PLNCOL     ;planet color
        LDA INIBVL,X   ;bomb value low
        STA BOMVL      ;save it
        LDA INIBVH,X   ;bomb value hi
        STA BOMVH      ;save it
        LDA INISV,X    ;saucer value
        STA SAUVAL     ;save that too
        CPX #11        ;at level 11?
        BEQ SAMLVL     ;Yes. skip next
        INC BLEVEL     ;inc bomb level
SAMLVL  SED            ;decimal mode
        LDA LEVEL      ;game level #
        CLC            ;clear carry
        ADC #1         ;add one
        STA LEVEL      ;save game level
        CLD            ;clear decimal
;
; -----------------
; Main program loop
; -----------------
;
LOOP    LDA PAUSED     ;game paused?
        BNE LOOP       ;Yes. loop here
        LDA #0         ;get zero
```

```
        STA ATRACT     ;attract mode
        LDA GAMCTL     ;game done?
        BPL CKCORE     ;No. check core
        LDA EXPCNT     ;Yes. expl count
        BNE CKCORE     ;count done? No.
        JMP ENDGAM     ;The End!
;
; --------------------------
; Check planet core for hit!
; --------------------------
;
CKCORE  LDA SCRN+1939  ;center LF
        AND #$03       ;RT color clock
        CMP #$03       ;explosion colr?
        BEQ PLDEAD     ;Yes. go dead
        LDA SCRN+1940  ;center RT
        AND #$C0       ;LF color clock
        CMP #$C0       ;explosion colr?
        BNE PLANOK     ;No. skip next
;
; ---------------
; Planet is Dead!
; ---------------
;
PLDEAD  LDA #0         ;get zero
        STA BOMBS      ;zero bombs
        STA SATLIV     ;satellite dead
        LDA #$FF       ;get #$FF
        STA LIVES      ;no lives left
        STA GAMCTL     ;game control
        JSR SNDOFF     ;no sound
;
; -------------
; Check console
; -------------
;
PLANOK  LDA CONSOL     ;get console
        CMP #7         ;any pressed?
        BEQ NORST      ;No. skip next
        JMP PLANET     ;restart game!
;
; -----------------
; Projectile firing
; -----------------
;
NORST   JSR BOMINI     ;try new bomb
        LDA SATLIV     ;satellite stat
        BEQ NOTRIG     ;alive? No.
        LDA STRIG0     ;get trigger
        CMP LASTRG     ;same as last VB
        BEQ NOTRIG     ;Yes. skip next
        STA LASTRG     ;No. save trig
        CMP #0         ;pressed?
        BNE NOTRIG     ;No. skip next
        JSR PROINI     ;strt projectile
NOTRIG  JSR BOMADV     ;advance bombs
        LDA EXPTIM     ;do explosion?
        BNE NOEXP      ;no!
        JSR CHKSAT     ;satellite ok?
        JSR CHKHIT     ;any hits?
        JSR EXPHAN     ;handle expl.
        JSR PROADV     ;advance shots
        LDA SAUCER     ;saucer flag
        BEQ RESTIM     ;saucer? No.
        JSR SSHOOT     ;Yes. let shoot
RESTIM  LDA #1         ;get one
        STA EXPTIM     ;reset timer
NOEXP   LDA BOMBS      ;# bombs to go
        BNE LOOP       ;any left? Yes.
        LDA GAMCTL     ;game control
        BMI LOOP       ;dead? Yes.
        LDA BOMACT     ;bomb 0 status
        ORA BOMACT+1   ;bomb 1 status
        ORA BOMACT+2   ;bomb 2 status
        ORA BOMACT+3   ;bomb 3 status
        BEQ JSL        ;any bombs? No.
        JMP LOOP       ;Yes. continue
JSL     JMP SETLVL     ;setup new level
;
; ------------------------
; Initiate a new explosion
; ------------------------
;
NEWEXP  LDA #64        ;1.07 seconds
        STA EXSCNT     ;expl sound cnt
        INC EXPCNT     ;one more expl
        LDY EXPCNT     ;use as index
        LDA NEWX       ;put X coord
        STA XPOS,Y     ;into X table
        LDA NEWY       ;put Y coord
        STA YPOS,Y     ;into Y table
        LDA #0         ;init to zero
        STA CNT,Y      ;explosion image
RT1     RTS            ;return
;
; ------------------------------
; Main explosion handler routine
; ------------------------------
;
EXPHAN  LDA #0         ;init to zero
        STA COUNTR     ;zero counter
RUNLP   INC COUNTR     ;nxt explosion
        LDA EXPCNT     ;get explosion #
        CMP COUNTR     ;any more expl?
```

```
         BMI RT1      ;No. return
         LDX COUNTR   ;get index
         LDA #Ø       ;init plotclr
         STA PLOTCLR  ;Ø = plot block
         LDA CNT,X    ;expl counter
         CMP #37      ;all drawn?
         BMI DOPLOT   ;No. do it
         INC PLOTCLR  ;1 = erase block
         SEC          ;set carry
         SBC #37      ;erase cycle
         CMP #37      ;erase done?
         BMI DOPLOT   ;No. erase block
         TXA          ;move index
         TAY          ;to Y register
;
; ---------------------------
; Repack explosion table, get
; rid of finished explosions
; ---------------------------
;
REPACK   INX          ;next explosion
         CPX EXPCNT   ;done?
         BEQ RPK2     ;No. repack more
         BPL RPKEND   ;Yes. exit
RPK2     LDA XPOS,X   ;get X position
         STA XPOS,Y   ;move back X
         LDA YPOS,X   ;get Y position
         STA YPOS,Y   ;move back Y
         LDA CNT,X    ;get count
         STA CNT,Y    ;move back count
         INY          ;inc index
         BNE REPACK   ;next repack
RPKEND   DEC EXPCNT   ;dec pointers
         DEC COUNTR   ;due to repack
         JMP RUNLP    ;continue
DOPLOT   INC CNT,X    ;inc pointer
         TAY          ;exp phase in Y
         LDA XPOS,X   ;get X-coord
         CLC          ;clear carry
         ADC COORD1,Y ;add X offset
         STA PLOTX    ;save it
         CMP #16Ø     ;off screen?
         BCS RUNLP    ;Yes. don't plot
         LDA YPOS,X   ;get Y-coord
         ADC COORD2,Y ;add Y offset
         STA PLOTY    ;save it
         CMP #96      ;off screen?
         BCS RUNLP    ;Yes. don't plot
         JSR PLOT     ;get plot addr
         LDA PLOTCLR  ;erase it?
         BNE CLEARIT  ;Yes. clear it
         LDA PLOTBL,X ;get plot bits
         ORA (LO),Y   ;alter display
PUTIT    STA (LO),Y   ;and replot it!
         JMP RUNLP    ;exit
CLEARIT  LDA ERABIT,X ;erase bits
         AND (LO),Y   ;turn off pixel
         JMP PUTIT    ;put it back
;
; ------------------------
; Dedicated multiply by 4Ø
; with result in LO and HI
; ------------------------
;
PLOT     LDA PLOTY    ;get Y-coord
         ASL A        ;shift it left
         STA LO       ;save low *2
         LDA #Ø       ;get zero
         STA HI       ;init high byte
         ASL LO       ;shift low byte
         ROL HI       ;rotate high *4
         ASL LO       ;shift low byte
         LDA LO       ;get low byte
         STA LOHLD    ;save low *8
         ROL HI       ;rotate high *8
         LDA HI       ;get high byte
         STA HIHLD    ;save high *8
         ASL LO       ;shift low byte
         ROL HI       ;rotate high *16
         ASL LO       ;shift low byte
         ROL HI       ;rotate high *32
         LDA LO       ;get low *32
         CLC          ;clear carry
         ADC LOHLD    ;add low *8
         STA LO       ;save low *4Ø
         LDA HI       ;get high *32
         ADC HIHLD    ;add high *8
         STA HI       ;save high *4Ø
;
; -----------------------------
; Get offset into screen memory
; -----------------------------
;
         LDA # <SCRN  ;screen addr lo
         CLC          ;clear carry
         ADC LO       ;add low offset
         STA LO       ;save addr low
         LDA # >SCRN  ;screen addr hi
         ADC HI       ;add high offset
         STA HI       ;save addr hi
         LDA PLOTX    ;mask PLOTX for
         AND #3       ;the plot bits,
         TAX          ;place in X..
         LDA PLOTX    ;get PLOTX and
         LSR A        ;divide
         LSR A        ;by 4
         CLC          ;and add to
         ADC LO       ;plot address
         STA LO       ;for final plot
         BCC PLOT1    ;address.
         INC HI       ;overflow? Yes.
PLOT1    LDY #Ø       ;zero Y register
         RTS          ;return
;
; ---------------
; Bomb initializer
; ---------------
;
BOMINI   LDA BOMBWT   ;bomb wait time
         BNE NOBINI   ;done? No.
         LDA BOMBS    ;more bombs?
         BNE CKLIVE   ;Yes. skip RTS
NOBINI   RTS          ;No. return
CKLIVE   LDX #3       ;find..
CKLVLP   LDA BOMACT,X ;an available..
         BEQ GOTBOM   ;bomb? Yes.
         DEX          ;No. dec index
         BPL CKLVLP   ;done? No.
         RTS          ;return
GOTBOM   LDA #1       ;this one is..
         STA BOMACT,X ;active now
         DEC BOMBS    ;one less bomb
         LDA #Ø       ;zero out all..
         STA BXHOLD,X ;vector X hold
         STA BYHOLD,X ;vector Y hold
         LDA GAMCTL   ;game control
         BMI NOSAUC   ;saucer possible?
;
; --------------
; Saucer handler
; --------------
;
         CPX #3       ;Yes. bomb #3?
         BNE NOSAUC   ;No. skip next
         LDA RANDOM   ;random number
         CMP SAUCHN   ;compare chances
         BCS NOSAUC   ;put saucer? No.
         LDA #1       ;Yes. get one
         STA SAUCER   ;enable saucer
         LDA RANDOM   ;random number
         AND #$Ø3     ;range: Ø..3
         TAY          ;use as index
         LDA STARTX,Y ;saucer start X
         CMP #$FF     ;random flag?
         BNE SAVESX   ;No. use as X
         JSR SAURND   ;random X-coord
         ADC #35      ;add X offset
SAVESX   STA FROMX    ;from X vector
         STA BOMBX,X  ;init X-coord
         LDA STARTY,Y ;saucer start Y
         CMP #$FF     ;random flag?
         BNE SAVESY   ;No. use as Y
         JSR SAURND   ;random Y-coord
         ADC #55      ;add Y offset
SAVESY   STA FROMY    ;from Y vector
         STA BOMBY,X  ;init Y-coord
         LDA ENDX,Y   ;saucer end X
         CMP #$FF     ;random flag?
         BNE SAVEEX   ;No. use as X
         LDA #23Ø     ;screen right
         SEC          ;offset so not
         SBC FROMY    ;to hit planet
SAVEEX   STA TOX      ;to X vector
         LDA ENDY,Y   ;saucer end Y
         CMP #$FF     ;random flag?
         BNE SAVEEY   ;No. use as Y
         LDA FROMX    ;use X for Y
SAVEEY   STA TOY      ;to Y vector
         JMP GETBV    ;skip next
;
; ------------
; Bomb handler
; ------------
;
NOSAUC   LDA RANDOM   ;random number
         BMI BXMAX    ;coin flip
         LDA RANDOM   ;random number
         AND #1       ;make Ø..1
         TAY          ;use as index
         LDA BMAXS,Y  ;top/bottom tbl
         STA BOMBY,X  ;bomb Y-coord
SETRBX   LDA RANDOM   ;random number
         CMP #25Ø     ;compare w/25Ø
         BCS SETRBX   ;less than? No.
         STA BOMBX,X  ;bomb X-coord
         JMP BOMVEC   ;skip next
BXMAX    LDA RANDOM   ;random number
         AND #1       ;make Ø..1
         TAY          ;use as index
         LDA BMAXS,Y  ;Ø or 25Ø
         STA BOMBX,X  ;bomb X-coord
SETRBY   LDA RANDOM   ;random number
         CMP #25Ø     ;compare w/25Ø
         BCS SETRBY   ;less than? No.
         STA BOMBY,X  ;bomb Y-coord
BOMVEC   LDA BOMBX,X  ;bomb X-coord
         STA FROMX    ;shot from X
         LDA BOMBY,X  ;bomb Y-coord
         STA FROMY    ;shot from Y
         LDA #128     ;planet center
         STA TOX      ;shot to X-coord
         STA TOY      ;shot to Y-coord
GETBV    JSR VECTOR   ;calc shot vect
;
; ---------------------
; Store vector in table
; ---------------------
;
         LDA LR       ;bomb L/R flag
         STA BOMBLR,X ;bomb L/R table
         LDA UD       ;bomb U/D flag
         STA BOMBUD,X ;bomb U/D table
         LDA VXINC    ;velocity X inc
         STA BXINC,X  ;Vel X table
         LDA VYINC    ;velocity Y inc
         STA BYINC,X  ;Vel Y table
         RTS          ;return
;
; ------------------------------
; Saucer random generator Ø..99
; ------------------------------
;
SAURND   LDA RANDOM   ;random number
         AND #$7F     ;Ø..127
         CMP #1ØØ     ;compare w/1ØØ
         BCS SAURND   ;less than? No.
         RTS          ;return
;
; --------------------
; Saucer shoot routine
; --------------------
;
SSHOOT   LDA RANDOM   ;random number
         CMP #6       ;2.3% chance?
         BCS NOSS     ;less than? No.
         LDX #7       ;7 = index
         LDA PROACT,X ;projectile #7
         BEQ GOTSS    ;active? No.
         DEX          ;6 = index
         LDA PROACT,X ;projectile #6
         BEQ GOTSS    ;active? No.
NOSS     RTS          ;return, no shot
;
; --------------------
; Enable a saucer shot
; --------------------
;
GOTSS    LDA #48      ;PF center, Y
         STA TOY      ;shot to Y-coord
         LDA #8Ø      ;PF center X
         STA TOX      ;shot to X-coord
         LDA BOMBX+3  ;saucer x-coord
         SEC          ;set carry
         SBC #44      ;PF offset
         STA FROMX    ;shot from X
         STA PROJX,X  ;X-coord table
         CMP #16Ø     ;screen X limit
         BCS NOSS     ;on screen? No.
         LDA BOMBY+3  ;saucer Y-coord
         SBC #37      ;PF offset
         LSR A        ;2 scan lines
         STA FROMY    ;shot from Y
         STA PROJY,X  ;Y-coord table
         CMP #95      ;screen Y limit
         BCS NOSS     ;on screen? No.
         LDA #13      ;shot snd time
         STA ESSCNT   ;emeny snd count
         JMP PROVEC   ;continue
;
; ----------------------
; Projectile initializer
; ----------------------
;
PROINI   LDX #5       ;6 projectiles
PSCAN    LDA PROACT,X ;get status
         BEQ GOTPRO   ;active? No.
         DEX          ;Yes. try again
         BPL PSCAN    ;done? No.
         RTS          ;return
;
; ----------------
; Got a projectile!
; ----------------
;
GOTPRO   LDA #13      ;shot snd time
         STA PSSCNT   ;player sht snd
         LDA SATX     ;satellite X
         STA FROMX    ;shot from X
         STA PROJX,X  ;proj X table
         LDA SATY     ;satellite Y
         STA FROMY    ;shot from Y
         STA PROJY,X  ;proj Y table
         LDA CURX     ;cursor X-coord
         SEC          ;set carry
         SBC #48      ;playfld offset
         STA TOX      ;shot to X-coord
         LDA CURY     ;cursor Y-coord
         SEC          ;set carry
         SBC #32      ;playfld offset
```

```
        LSR A         ;2 line res
        STA TOY       ;shot to Y-coord
PROVEC  JSR VECTOR    ;compute vect
        LDA VXINC     ;X increment
        STA PXINC,X   ;X inc table
        LDA VYINC     ;Y increment
        STA PYINC,X   ;Y inc table
        LDA LR        ;L/R flag
        STA PROJLR,X  ;L/R flag table
        LDA UD        ;U/D flag
        STA PROJUD,X  ;U/D flag table
        LDA #1        ;active
        STA PROACT,X  ;proj status
RT2     RTS           ;return
;
; --------------------
; Bomb advance handler
; --------------------
;
BOMADV  LDA BOMTIM    ;bomb timer
        BNE RT2       ;time up? No.
        LDA LIVES     ;any lives?
        BPL REGBT     ;Yes. skip next
        LDA #1        ;speed up bombs
        BNE SETBTM    ;skip next
REGBT   LDA BOMTI     ;get bomb speed
SETBTM  STA BOMTIM    ;reset timer
        LDX #3        ;check 4 bombs
ADVBLP  LDA BOMACT,X  ;bomb on?
        BEQ NXTBOM    ;No. try next
        JSR ADVIT     ;advance bomb
        LDA LIVES     ;any lives left?
        BPL SHOBOM    ;Yes. skip next
        JSR ADVIT     ;No. move bombs
        JSR ADVIT     ;4 times faster
        JSR ADVIT     ;than normal
;
; --------------------------
; We've now got updated bomb
; coordinates for plotting!
; --------------------------
;
SHOBOM  LDA BOMBY,X   ;bomb Y-coord
        CLC           ;clear carry
        ADC #2        ;bomb center off
        STA INDX1     ;save it
        LDA #Ø        ;get zero
        STA LO        ;init low byte
        TXA           ;index to Acc
        ORA # >PLRØ   ;mask w/address
        STA HI        ;init high byte
        STX INDX2     ;X temp hold
        CPX #3        ;saucer slot?
        BNE NOTSAU    ;No. skip next
        LDA SAUCER    ;saucer in slot?
        BNE NXTBOM    ;Yes. skip bomb
NOTSAU  LDY BOMBLR,X  ;L/R flag
        LDA #17       ;do 17 bytes
        STA TEMP      ;set counter
        LDX BPSTRT,Y  ;start position
        LDY INDX1     ;bomb Y pos
BDRAW   CPY #32       ;off screen top?
        BCC NOBDRW    ;Yes. skip next
        CPY #223      ;screen bottom?
        BCS NOBDRW    ;Yes. skip next
        LDA BOMPIC,X  ;bomb picture
        STA (LO),Y    ;put in PM area
NOBDRW  DEY           ;PM index
        DEX           ;picture index
        DEC TEMP      ;dec count
        BNE BDRAW     ;done? No.
        LDX INDX2     ;restore X
        LDA BOMBX,X   ;bomb X-coord
        STA HPOSPØ,X  ;player pos
NXTBOM  DEX           ;more bombs?
        BPL ADVBLP    ;yes!
        RTS           ;all done!
;
; --------------------------
; Projectile advance handler
; --------------------------
;
PROADV  LDX #11       ;do 8: 11..4
PADVLP  LDA BOMACT,X  ;active?
        BEQ NXTPRO    ;No. skip next
        LDA BOMBX,X   ;bomb X-coord
        STA PLOTX     ;plotter X
        LDA BOMBY,X   ;bomb Y-coord
        STA PLOTY     ;plotter Y
        STX XHOLD     ;X-reg temporary
        JSR PLOT      ;calc plot addr
        LDA (LO),Y    ;get plot byte
        AND ERABIT,X  ;erase bit
        STA (LO),Y    ;replace byte
        LDX XHOLD     ;restore X
        JSR ADVIT     ;advance proj
        LDA BOMBX,X   ;bomb X-coord
        CMP #16Ø      ;off screen?
        BCS KILPRO    ;Yes. kill it
        STA PLOTX     ;plotter X
        LDA BOMBY,X   ;bomb Y-coord
        CMP #96       ;off screen?
        BCS KILPRO    ;Yes. kill it
        STA PLOTY     ;plotter Y
        JSR PLOT      ;calc plot addr
        LDA PLOTBL,X  ;get plot mask
        AND (LO),Y    ;chk collision
        BEQ PROJOK    ;No. plot it
        LDX XHOLD     ;restore X
        LDA PLOTX     ;proj X-coord
        STA NEWX      ;explo X-coord
        LDA PLOTY     ;proj Y-coord
        STA NEWY      ;explo Y-coord
        JSR NEWEXP    ;set off explo
KILPRO  LDA #Ø        ;get zero
        STA BOMACT,X  ;kill proj
        JMP NXTPRO    ;skip next
PROJOK  LDA PLOTBL,X  ;plot mask
        LDX XHOLD     ;restore X
        AND PROMSK,X  ;mask color
        ORA (LO),Y    ;add playfield
        STA (LO),Y    ;replace byte
NXTPRO  DEX           ;next projectile
        CPX #3        ;proj #3 yet?
        BNE PADVLP    ;No. continue
        RTS           ;return
;

;
; File: D:PLANET3.M65
; ----------------------
; Check satellite status
; ----------------------
;
CHKSAT  LDA DEADTM    ;satellite ok?
        BEQ LIVE      ;No. skip next
CHKSX   RTS           ;return
LIVE    LDA LIVES     ;lives left?
        BMI CHKSX     ;No. exit
        LDA #1        ;get one
        STA SATLIV    ;set alive flag
        LDA MØPL      ;did satellite
        ORA MØPL+1    ;hit any bombs?
        BEQ CHKSX     ;No. exit
        LDA #Ø        ;get zero
        STA SATLIV    ;kill satellite
        STA SCNT      ;init orbit
        LDX LIVES     ;one less life
        STA SCOLIN+14,X ;erase life
        DEC LIVES     ;dec lives count
        BPL MORSAT    ;any left? Yes.
        LDA #255      ;lot of bombs
        STA BOMBS     ;into bomb count
        STA GAMCTL    ;end game
        JSR SNDOFF    ;no sound 1 2 3
MORSAT  LDA SATX      ;sat X-coord
        STA NEWX      ;explo X-coord
        LDA SATY      ;sat Y-coord
        STA NEWY      ;explo Y-coord
        JSR NEWEXP    ;set off explo
        LDA #8Ø       ;init sat X
        STA SATX      ;sat X-coord
        LDA #21       ;init sat Y
        STA SATY      ;sat Y-coord
        LDX #Ø        ;don't show the
CLRSAT  LDA MISL,X    ;satellite pic
        AND #$FØ      ;mask off sat
        STA MISL,X    ;restore data
        DEX           ;dec index
        BNE CLRSAT    ;done? No.
        LDA #$FF      ;4.25 seconds
        STA DEADTM    ;till next life!
        RTS           ;return
;
; ------------------
; Check console keys
; ------------------
;
ENDGAM  JSR SNDOFF    ;no sound 123
ENDGLP  LDA STRIGØ    ;stick trigger
        AND PTRIGØ    ;mask w/paddle Ø
        AND PTRIG1    ;mask w/paddle 1
        BEQ ENDGL1    ;any pushed? No.
        LDA CONSOL    ;chk console
        CMP #7        ;any pushed?
        BEQ ENDGLP    ;No. loop here
ENDGL1  JMP PLANET    ;restart game
;
; -------------------------
; Turn off sound regs 1 2 3
; -------------------------
;
SNDOFF  LDA #Ø        ;zero volume
        STA AUDC1     ;to sound #1
        STA AUDC1+2   ;sound #2
        STA AUDC1+4   ;sound #3
        RTS           ;return
;
; -----------------------
; Check for hits on bombs
; -----------------------
;
CHKHIT  LDX #3        ;4 bombs Ø..3
        LDA SAUCER    ;saucer enabled?
        BEQ CHLOOP    ;No. skip next
        LDA #Ø        ;get zero
        STA BOMCOL    ;collision count
        LDA GAMCTL    ;game over?
        BMI NOSCOR    ;Yes. skip next
        LDA BOMBX+3   ;saucer X-coord
        CMP #39       ;off screen lf?
        BCC NOSCOR    ;Yes. kill it
        CMP #211      ;off screen rt?
        BCS NOSCOR    ;Yes. kill it
        LDA BOMBY+3   ;saucer Y-coord
        CMP #19       ;off screen up?
        BCC NOSCOR    ;Yes. kill it
        CMP #231      ;off screen dn?
        BCS NOSCOR    ;Yes. kill it
CHLOOP  LDA #Ø        ;get zero
        STA BOMCOL    ;collision count
        LDA PØPF,X    ;playf collision
        AND #$Ø5      ;w/shot+planet
        BEQ NOBHIT    ;hit either? No.
        INC BOMCOL    ;Yes. inc count
        AND #$Ø4      ;hit shot?
        BEQ NOSCOR    ;No. skip next
        LDA GAMCTL    ;game over?
        BMI NOSCOR    ;Yes. skip next
        LDA #2        ;1/3Øth second
        STA BOMBWT    ;bomb wait time
        CPX #3        ;saucer player?
        BNE ADDBS     ;No. skip this
        LDA SAUCER    ;saucer on?
        BEQ ADDBS     ;No. this this
        LDA SAUVAL    ;saucer value
        STA SCOADD+1  ;point value
        JMP ADDIT     ;add to score
;
; -----------------------
; Add bomb value to score
; -----------------------
;
ADDBS   LDA BOMVL     ;bomb value low
        STA SCOADD+2  ;score inc low
        LDA BOMVH     ;bomb value high
        STA SCOADD+1  ;score inc high
ADDIT   STX XHOLD     ;save X register
        JSR ADDSCO    ;add to score
        LDX XHOLD     ;restore X
NOSCOR  LDA #Ø        ;get zero
        STA BOMACT,X  ;kill bomb
        LDY BOMBLR,X  ;L/R flag
        LDA BOMBX,X   ;bomb X-coord
        SEC           ;set carry
        SBC BXOF,Y    ;bomb X offset
        STA NEWX      ;plotter X-coord
        LDA BOMBY,X   ;bomb Y-coord
        SEC           ;set carry
        SBC #4Ø       ;bomb Y offset
        LSR A         ;2 line res.
        STA NEWY      ;plotter Y-coord
        LDA SAUCER    ;saucer?
        BEQ EXPBOM    ;No. explode it
        CPX #3        ;bomb player?
        BNE EXPBOM    ;Yes. explode it
        LDA #Ø        ;get zero
        STA SAUCER    ;kill saucer
        JSR CLRPLR    ;clear player
        LDA GAMCTL    ;game over?
        BMI NOBHIT    ;Yes. skip next
EXPBOM  JSR CLRPLR    ;clear player
        LDA BOMCOL    ;collisions?
        BEQ NOBHIT    ;No. skip this
        JSR NEWEXP    ;init explosion
NOBHIT  DEX           ;dec index
        BPL CHLOOP    ;done? No.
        STA HITCLR    ;reset collision
        RTS           ;return
;
; -------------------------
; Advance bombs/projectiles
; -------------------------
;
ADVIT   LDA BXHOLD,X  ;bomb X-sum
        CLC           ;clear carry
        ADC BXINC,X   ;add X-increment
        STA BXHOLD,X  ;replace X-sum
        LDA #Ø        ;get zero
        ROL A         ;carry = 1
        STA DELTAX    ;X-delta
        LDA BYHOLD,X  ;bomb Y-sum
        ADC BYINC,X   ;add Y-increment
        STA BYHOLD,X  ;replace Y-sum
        LDA #Ø        ;get zero
        ROL A         ;carry = 1
        STA DELTAY    ;Y-delta
        LDA BOMBLR,X  ;bomb L/R flag
        BEQ ADVLFT    ;go left? Yes.
        LDA BOMBX,X   ;bomb X-coord
        ADC DELTAX    ;add X-delta
        JMP ADVY      ;skip next
ADVLFT  LDA BOMBX,X   ;bomb X-coord
        SEC           ;set carry
        SBC DELTAX    ;sub X-delta
```

```
ADVY     STA BOMBX,X ;save X-coord
         LDA BOMBUD,X ;bomb U/D flag
         BEQ ADVDN    ;go down? Yes.
         LDA BOMBY,X ;bomb Y-coord
         SEC          ;set carry
         SBC DELTAY   ;sub Y-delta
         JMP ADVEND   ;skip next
ADVDN    LDA BOMBY,X ;bomb Y-coord
         CLC          ;clear carry
         ADC DELTAY   ;add Y-delta
ADVEND   STA BOMBY,X ;save Y-coord
         RTS          ;return
;
; -------------------------
; Clear out player indicated
; by the X register!
; -------------------------
;
CLRPLR   LDA #0       ;move player...
         STA HPOSP0,X ;off screen,
         TAY          ;init index
         TXA          ;get X
         ORA # >PLR0  ;mask w/address
         STA HI       ;plr addr high
         TYA          ;Acc = 0
         STA LO       ;plr addr low
CLPLP    STA (LO),Y   ;zero player
         DEY          ;dec index
         BNE CLPLP    ;done? No.
         RTS          ;return
;
; ----------------------
; Calculate target vector
; ----------------------
;
VECTOR   LDA #0       ;get zero
         STA LR       ;going left
         LDA FROMX    ;from X-coord
         CMP TOX      ;w/to X-coord
         BCC RIGHT    ;to right? Yes.
         SBC TOX      ;get X-diff
         JMP VECY     ;skip next
RIGHT    INC LR       ;going right
         LDA TOX      ;to X-coord
         SEC          ;set carry
         SBC FROMX    ;get X-diff
VECY     STA VXINC    ;save difference
         LDA #1       ;get one
         STA UD       ;going up flag
         LDA FROMY    ;from Y-coord
         CMP TOY      ;w/to Y-coord
         BCC DOWN     ;down? Yes.
         SBC TOY      ;get Y-diff
         JMP VECSET   ;skip next
DOWN     DEC UD       ;going down flag
         LDA TOY      ;to Y-coord
         SEC          ;set carry
         SBC FROMY    ;get Y-diff
VECSET   STA VYINC    ;are both
         ORA VXINC    ;distances 0?
         BNE VECLP    ;No. skip next
         LDA #$80     ;set x increment
         STA VXINC    ;to default.
VECLP    LDA VXINC    ;X vector incre
         BMI VECEND   ;>127? Yes.
         LDA VYINC    ;Y vector incre
         BMI VECEND   ;>127? Yes.
         ASL VXINC    ;times 2 until
         ASL VYINC    ;one is >127
         JMP VECLP    ;continue
VECEND   RTS          ;return
;
; ------------
; Add to score
; ------------
;
ADDSCO   LDY #0       ;init index
         SED          ;decimal mode
         CLC          ;clear carry
         LDX #2       ;do 3 bytes
ASCLP    LDA SCORE,X ;get score
         ADC SCOADD,X ;add bomb value
         STA SCORE,X ;save score
         STY SCOADD,X ;zero value
         DEX          ;next byte
         BPL ASCLP    ;done? No.
         CLD          ;clear decimal
;
; ----------
; Show score
; ----------
;
SHOSCO   LDA #$10     ;put color 0
         STA SHCOLR   ;in hold area
         LDX #1       ;2nd line char
         LDY #0       ;digits 1,2
SSCOLP   LDA SCORE,Y ;get digits
         JSR SHOBCD   ;show 'em
         INX          ;advance score
         INX          ;line pointer
         INY          ;next 2 digits
         CPY #3       ;done 6?
         BNE SSCOLP   ;no!
         RTS          ;all done!
;
; -----------------
; Show level number
; -----------------
;
SHOLVL   LDY #$50     ;use color 2
         STY SHCOLR   ;save it
         LDA LEVEL    ;get level #
         LDX #11      ;12th char on line
;
; -----------------
; Show 2 BCD digits
; -----------------
;
SHOBCD   STA SHOBYT   ;save digits
         AND #$0F     ;get lower digit
         ORA SHCOLR   ;add color
         STA SCOLIN+1,X ;show it
         LDA SHOBYT   ;get both again
         LSR A        ;mask...
         LSR A        ;off...
         LSR A        ;upper...
         LSR A        ;digit
         ORA SHCOLR   ;add color
         STA SCOLIN,X ;show it!
         RTS          ;and exit.
;
; ------------------
; KOALA PAD interface
; ------------------
;
; The following filtering
; algorithm is used:
;
; Given 5 points S1,S2,S3,S4,S5
;
; R1=S1+S2+S2+S3
; R2=S2+S3+S3+S4
; R3=S3+S4+S4+S5
;
; AVG=(R1+R2+R2+R3)/16
;
; This reduces to:
;
; AVG=(S1+S2*4+S3*6+S4*4+S5)/16
;
; --------------------------
; Rotate points through queue
; --------------------------
;
KOALA    LDX #4       ;do 5 bytes
ROT      LDA XQ-1,X   ;move X queue
         STA XQ,X     ;up one byte
         LDA YQ-1,X   ;move Y queue
         STA YQ,X     ;up one byte
         DEX          ;dec count
         BNE ROT      ;done? No.
;
; -------------------
; Clear out the cursor
; -------------------
;
         LDY CURY     ;get Y coord
         LDX #5       ;do 6 bytes
CCURS    LDA MISL,Y   ;get missiles
         AND #$F0     ;mask off low
         STA MISL,Y   ;put back
         DEX          ;dec count
         BPL CCURS    ;done? No.
;
; ------------------------
; Insert new point into queue
; ------------------------
;
         LDA #1       ;pen up flag
         STA PENFLG   ;set pen up
         LDA PADDL0   ;X input
         STA XQ       ;put in queue
         CMP #5       ;screen boundary
         BCC KOALAX   ;on screen? No.
         LDA PADDL1   ;Y input
         STA YQ       ;put in queue
         CMP #5       ;screen boundary
         BCC KOALAX   ;on screen? No.
;
; --------------------
; Filter the X-Y queues
; --------------------
;
         LDA # <XQ    ;queue addr low
         STA PTR      ;pointer low
         LDA # >XQ    ;queue addr high
         STA PTR+1    ;pointer high
         JSR FILTER   ;filter X data
         BCS KOALAX   ;good data? No.
         ADC #16      ;X offset
         CMP #48      ;far left?
         BCS FLF      ;No. skip
         LDA #48      ;screen left
FLF      CMP #208     ;far right?
         BCC FRT      ;No. skip
         LDA #207     ;screen right
FRT      STA CURX     ;put X coord
         LDA # <YQ    ;queue addr low
         STA PTR      ;pointer low
         LDA # >YQ    ;queue addr high
         STA PTR+1    ;pointer high
         JSR FILTER   ;filter Y data
         BCS KOALAX   ;good data? No.
         ADC #16      ;Y offset
         CMP #32      ;above top?
         BCS FUP      ;No. skip
         LDA #32      ;screen top
FUP      CMP #224     ;below bottom?
         BCC FDN      ;No. skip
         LDA #223     ;screen bottom
FDN      STA CURY     ;put Y coord
;
; ----------------------
; Paddle trigger handler
; ----------------------
;
         LDA PTRIG0   ;paddle trig 0
         EOR PTRIG1   ;EOR w/PTRIG1
         EOR #1       ;inverse data
         STA STRIG0   ;put in STRIG0
         LDA #0       ;pen down flag
         STA PENFLG   ;set pen down
KOALAX   RTS          ;continue
;
; ---------------------------
; Filter algorithm, initialize
; ---------------------------
;
FILTER   LDA #0       ;get zero
         LDX #4       ;do 5 bytes
FILC     STA SH,X     ;high byte table
         DEX          ;dec count
         BPL FILC     ;done? No.
         STA AVG      ;average low
         STA AVG+1    ;average high
         TAY          ;zero in Y
         LDX #1       ;one in X
;
; ----------------------
; Process the X-Y samples
; ----------------------
;
         LDA (PTR),Y  ;get S1
         STA SL,Y     ;save low byte
         INY          ;inc pointer
         JSR MUL4     ;process S2
         LDA (PTR),Y  ;get S3
         ASL A        ;times 2
         ROL SH,X     ;rotate carry
         ADC (PTR),Y  ;add = times 3
         BCC FIL2     ;overflow? No.
         INC SH,X     ;inc high byte
FIL2     ASL A        ;times 2
         ROL SH,X     ;rotate carry
         STA SL,X     ;save low byte
         INX          ;inc pointer
         INY          ;inc pointer
         JSR MUL4     ;process S4
         LDA (PTR),Y  ;get S5
         STA SL,Y     ;save low byte
;
; -------------
; Total samples
; -------------
;
         LDX #4       ;add 5 elements
ALOOP    LDA SL,X     ;get low byte
         ADC AVG      ;add to average
         STA AVG      ;save low byte
         LDA SH,X     ;get high byte
         ADC AVG+1    ;add to average
         STA AVG+1    ;save high byte
         DEX          ;dec pointer
         BPL ALOOP    ;done? No.
;
; -----------------
; Divide total by 16
; -----------------
;
         LDX #4       ;shift 4 bits
         LDA AVG      ;get lo byte
DIV16    LSR AVG+1    ;rotate high
         ROR A        ;rotate low
         DEX          ;dec count
         BNE DIV16    ;done? No.
         TAX          ;save Acc
;
; ---------------------------
; Compare average with DELTA
; ---------------------------
;
MEAN     LDY #4       ;5 byte table
         SEC          ;set carry
         SBC (PTR),Y  ;compare points
         BCS POSI     ;negative? No.
         EOR #$FF     ;negate byte and
         ADC #1       ;+1 = ABS value
POSI     CMP #24      ;within DELTA?
         BCS FAIL     ;No. abort
         TXA          ;get Acc again
         DEY          ;dec pointer
```

```
        BPL MEAN     ;done? No.
FAIL    RTS          ;exit
;
; ----------------
; Multply Acc by 4
; ----------------
;
MUL4    LDA (PTR),Y  ;get S2
        ASL A        ;times 2
        ROL SH,X     ;rotate carry
        ASL A        ;times 4
        ROL SH,X     ;rotate carry
        STA SL,X     ;save low byte
        INX          ;inc pointer
        INY          ;inc pointer
        RTS          ;return
;
; ----------
; Data areas
; ----------
;
BMAXS   .BYTE 0,250 ;bomb limits
BOMPIC  .BYTE 0,0,0,0,0,0,$DC,$3E
        .BYTE $7E,$3E,$DC,0,0,0,0
        .BYTE 0,0,$76,$F8,$FC
        .BYTE $F8,$76,0,0,0,0,0,0
BPSTRT  .BYTE 27,16
BXOF    .BYTE 47,42
CURPIC  .BYTE $40,$40,$A0,$A0
        .BYTE $40,$40
P3COLR  .BYTE $34,$F8
SAUPIC  .BYTE 0,0,$18,$7E,0,0
        .BYTE $7E,$18,0,0
SAUMID  .BYTE $92,$49,$24,$92
STARTX  .BYTE 40,$FF,210,$FF
STARTY  .BYTE $FF,20,$FF,230
ENDX    .BYTE $FF,210,$FF,40
ENDY    .BYTE 20,$FF,230,$FF
;
; ---------------------
; Explosion data tables
; ---------------------
;
PLOTBL  .BYTE $C0,$30,$0C,$03
ERABIT  .BYTE $3F,$CF,$F3,$FC
PROMSK  .BYTE 0,0,0,0
        .BYTE $FF,$FF,$FF,$FF
        .BYTE $FF,$FF,$AA,$AA
COORD1  .BYTE 0,1,255,0,255,1
        .BYTE 0,2,255,254,0,1
        .BYTE 0,254,2,1,1,255
        .BYTE 0,2,254,255,3,0
        .BYTE 253,254,3,2,255,254
        .BYTE 1,255,3,253,1,253,2
COORD2  .BYTE 0,0,1,255,0,1
        .BYTE 1,0,255,1,2,255
        .BYTE 254,255,1,2,254,2
        .BYTE 3,255,0,254,1,253
        .BYTE 0,254,255,2,3,2
        .BYTE 253,253,0,1,3,255,254
;
; ------------------
; Initial score line
; ------------------
;
SCOINI  .BYTE $00,$00,$00,$00
        .BYTE $00,$00,$00,$00
        .BYTE $6C,$76,$6C,$00
        .BYTE $00,$00,$CA,$CA
        .BYTE $CA,$CA,$CA,$00
;
; ------------
; Level tables
; ------------
;
INIBOM  .BYTE 10,15,20,25,20,25
        .BYTE 15,20,25,20,25,30
INIBS   .BYTE 12,11,10,9,8,7
        .BYTE 7,6,5,5,4,3
INISC   .BYTE 0,10,50,90,50,80
        .BYTE 40,60,100,80,120,125
INIPC   .BYTE $20,$30,$40,$50,$60
        .BYTE $70,$80,$A0,$B0,$C0
        .BYTE $D0,$FF
INIBVH  .BYTE 0,0,0,0,0,0
        .BYTE 0,0,0,$01,$01,$01
INIBVL  .BYTE $10,$20,$30,$40,$50
        .BYTE $60,$70,$80,$90,$00
        .BYTE $10,$20
INISV   .BYTE 0,1,1,1,2,2
        .BYTE 3,3,3,4,4,4
;
; ----------
; Sound data
; ----------
;
PLSHOT  .BYTE 244,254,210,220
        .BYTE 176,186,142,152,108
        .BYTE 118,74,84,40,50
ENSHOT  .BYTE 101,96,85,80,69,64
        .BYTE 53,48,37,32,21,16,5,0
SAUSND  .BYTE 10,11,12,14,16,17
        .BYTE 18,17,16,14,12,11
```

```
;
; -----------------
; Program variables
; -----------------
;
XPOS    *=  *+20    ;all expl. x's
YPOS    *=  *+20    ;all expl. y's
CNT     *=  *+20    ;all expl. counts
BOMACT  *=  *+4     ;bomb active flags
PROACT  *=  *+8     ;proj. active flags
BOMBX   *=  *+4     ;bomb x positions
PROJX   *=  *+8     ;proj. x positions
BOMBY   *=  *+4     ;bomb y positions
PROJY   *=  *+8     ;proj. y positions
BXINC   *=  *+4     ;bomb x vectors
PXINC   *=  *+8     ;proj. x vectors
BYINC   *=  *+4     ;bomb y vectors
PYINC   *=  *+8     ;proj. y vectors
BXHOLD  *=  *+12    ;b/p hold areas
BYHOLD  *=  *+12    ;b/p hold areas
BOMBLR  *=  *+4     ;bomb left/right
PROJLR  *=  *+8     ;proj. left/right
BOMBUD  *=  *+4     ;bomb up/down
PROJUD  *=  *+8     ;proj. up/down
SCOLIN  *=  *+20    ;score line
;
; --------------
; End of program
; --------------
;
        .END
```

BOOT CAMP

An Assembly Language Tutorial Column.

by Tom Hudson

In this month's **Boot Camp**, we're going to finish our discussion of X and Y register indexing and become proficient in multi-byte addition.

Regular **Boot Camp** readers will be happy to know that the introductory material will be completely covered in the next few issues. After that, we can start applying all the 6502 instructions to useful subroutines and full-scale programs!

Solution #2.

I hope everyone at least tried to solve the indexing problem presented last issue. This problem asked readers to write the code necessary to copy the contents of the 6-byte TABLE1 to TABLE2 in reverse order. This little brain-teaser is an excellent opportunity to gain more experience with the 6502 index registers.

Figure 1 shows the code necessary to copy TABLE1 to TABLE2 in normal order. This figure was shown last month.

```
10        *=  $600
20        LDX #5
30 COPY   LDA TABLE1,X
40        STA TABLE2,X
50        DEX
60        BPL COPY
70        BRK
80 TABLE1 .BYTE 10,20,30,40,50,60
90 TABLE2 *=*+6
0100      .END
```

Figure 1.

I told you that only three changes to this code would allow it to copy the table in reverse order. The changed code is shown in **Figure 2**.

```
10        *=  $600
20        LDX #5
30        LDY #0
40 COPY   LDA TABLE1,X
50        STA TABLE2,Y
60        INY
70        DEX
80        BPL COPY
90        BRK
0100 TABLE1 .BYTE 10,20,30,40,50,60
0110 TABLE2 *=*+6
0120        .END
```

Figure 2.

How does it work? Let's step through the code and see.

Line 20 sets the X register to 5. This register will be used to point to different parts of TABLE1. With the index starting at 5, the register will point to the last byte of TABLE1.

Line 30 sets the Y register to 0. This register will be used to point to varying places in TABLE2. Unlike the X register, the Y register will start pointing at the *first* byte of TABLE2.

Lines 40-80 perform the table data move function.

Line 40 loads the accumulator with a byte from TABLE1, indicated by the X register.

Line 50 stores the byte just loaded into a byte of TABLE2, indicated by the Y register.

Lines 60 and 70 are the heart of this routine. Note that the Y register is INCREMENTED each time the BACKWD loop is executed, while the X register is DECREMENTED. **Figure 3** shows the X and Y register contents for each iteration of the loop.

```
TABLE1  TABLE2
 (X)     (Y)
------  ------
  5       0
  4       1
  3       2
  2       3
  1       4
  0       5
```

Figure 3.

By looking at **Figure 3**, you can see that the 6th byte (5+1) of TABLE1 will be moved to the 1st byte (0+1) of TABLE2, the 5th byte of TABLE1 to the 2nd byte of TABLE2, and so on.

Line 80 loops back to the BACKWD label if the X register is positive (0-127). Once the X register is decremented past 0, it "wraps around" to binary 1111111, or –1 decimal, and the program stops at the BRK instruction in line 90.

Line 100 sets up the initial values contained in TABLE1.

Line 110 tells the assembler to reserve 6 bytes for TABLE2. Remember, the "*=*+" directive allows you to set aside any number of bytes for tables, working areas, etc.

As a further example of the "reverse table" problem, **Figure 4** shows the BASIC equivalent of the assembly code in **Figure 2**.

```
10 DIM TABLE1(5),TABLE2(5)
15 TABLE1(0)=10:TABLE1(1)=20:TABLE1(2)
=30:TABLE1(3)=40:TABLE1(4)=50:TABLE1(5
)=60
20 X=5
30 Y=0
40 A=TABLE1(X)
50 TABLE2(Y)=A
60 Y=Y+1
70 X=X-1
80 IF X>=0 THEN 40
90 END
```

Figure 4.

Note that, in BASIC, it is necessary to initialize the TABLE1 array (line 15). This does the same thing as the .BYTE directive in line 100 of **Figure 2**.

This should give you a good idea of how indexing works. If you still have trouble, re-read last month's discussion of indexing and try developing your own simple problems.

Math revisited.

As promised last month, we're going to start looking at multi-byte math operations, both in binary and binary coded decimal (BCD).

Why do we want to bother with multi-byte math? If you're only working with numbers from 0-255, then single-byte math is fine. But what happens when you're writing the ultimate game program and need to show scores into the hundreds of thousands of points? Multi-byte math is the answer.

The simplest form of multi-byte math is probably two-byte address storage. The 6502 can address 65536 (or 2^{16}) bytes of memory. Observant readers will note that this number will easily fit into two eight-bit bytes.

You've probably encountered two-byte addresses in BASIC. For example, if you need to know where your computer's display list is located, you can use the BASIC command:

```
DLIST=PEEK(560)+PEEK(561)*256
```

How does this work? Normally, we think of a byte as having bit values from 1 to 128 (left to right). In order to represent larger numbers, we add a second *high order* byte to the first *low order* byte. The high order byte contains bit values from 2^8 (256) to 2^{15} (32768). This relationship is shown in **Figure 5**.

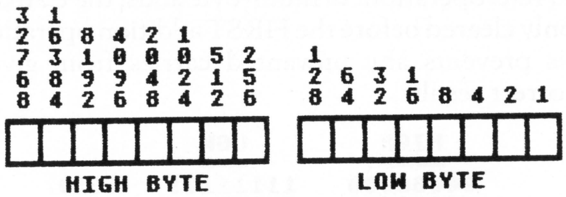


Figure 5.

In order for BASIC to reconstruct the number, it must multiply each byte by the value of its lowest-order bit. In the 2-byte case, the low-order byte is multiplied by 1, and the high-order byte is multiplied by 256. When the resulting numbers are added together, you have the value of the 2-byte number.

Figure 6 shows some decimal numbers along with their two-byte binary equivalents.

DECIMAL	HIGH BYTE	LOW BYTE
128	00000000	10000000
255	00000000	11111111
256	00000001	00000000
257	00000001	00000001
511	00000001	11111111
512	00000010	00000000
65534	11111111	11111110
0	00000000	00000000

Figure 6.

You don't have to stop with two bytes, either. For example, by using 3 bytes, you can store numbers up to 2^{24}, or 16,777,216. 4 bytes will give up to 2^{32}, or over 4 billion, and so on.

CARRYing on.

How is multi-byte math handled in 6502 assembly language? It's the same as single-byte, but with one difference. In multi-byte addition, the CARRY flag is used to handle carries and borrows.

You've used carries and borrows all your life, but you probably don't think about them. Consider the addition of 13+9. When you add 3+9, you get 12. Since 12 is greater than the maximum digit value of 9, you place the units portion (2) in the units position of the result and CARRY the 10 to the next digit. This adds to the tens digit of 13, giving 20. When this is added to the units portion calculated earlier, we get a result of 22.

In subtraction, if you're subtracting 7 from 20, 7 is larger than 0, so a borrow from the next digit is necessary. The 2 in the tens position becomes a 1, and the 7 is subtracted from the borrowed 10, giving a result of 3 in the units position. The final result is 13.

These same principles apply in multi-byte math operations. The only difference is the base we are operating in. As you recall from the Issue #15 **Boot Camp**, the CARRY flag is set to 1 if the result of an addition operation is greater than 255. In single-byte addition, we always CLEAR the CARRY flag before the ADC operation. In multi-byte adds, the CARRY **is only cleared before the FIRST addition operation.** This prevents any unwanted carries from giving incorrect results.

```
     HIGH          LOW
   --------      --------
   00000000      11111111   (255)
 + 00000000      00000001   (  1)
   --------      --------   -----
   00000001      00000000   (256)
```

Figure 7.

Figure 7 shows how carries work in binary. When 1 is added to 255, the resulting value of 256 is too large to fit in one byte. The low-order byte wraps around to zero and the carry flag is set. The high-order bytes are then added, along with the carry flag (1). This gives the high-order result a value of 1. Remember that the high-order byte of a two-byte value is always multiplied by 256. This gives us a final value of (1 X 256) + 0 = 256.

Figure 8 shows the code necessary for this addition operation in 6502 assembly code.

```
01      *=  $600
10      CLD              ;BINARY MODE
20      LDA #255         ;GET 255,
30      CLC              ;FIRST ADD!
40      ADC #1           ;ADD 1 TO 255
50      STA RESLO        ;STORE LOW RESULT
60      LDA #0           ;GET OP1 HIGH
70      ADC #0           ;ADD OP2 HIGH,
80      STA RESHI        ;SAVE HIGH RESULT
90      BRK              ;ALL DONE!
0100 RESLO *=*+1         ;LOW RESULT BYTE
0110 RESHI *=*+1         ;HIGH RESULT BYTE
0120   .END              ;END OF ASSEMBLY
```

Figure 8.

Line 10 clears the decimal mode, to make sure we're working with binary numbers.

Line 20 loads 255, the low byte of the first operand, into the accumulator.

Line 30 clears the carry flag for the first add operation. ALWAYS remember to clear the carry flag for the first add of a multi-byte add operation.

Line 40 adds 1, the low byte of the second operand, to the low byte of the first operand. This operation will leave a zero in the accumulator, and the carry flag will be set (1).

Line 50 stores the result of the low byte add in the location labeled RESLO.

Line 60 loads 0, the high byte of the first operand, into the accumulator.

Line 70 adds 0, the high byte of the first operand, to the high byte of the second operand. Note that we DID NOT clear the carry before this operation, since we want the carry status to be taken into account for all adds after the first one. In this case, with the carry set, our result is 0+0+1, or 1.

Line 80 stores the result of the high byte addition in the location labeled RESHI.

Line 90 stops the execution of the program with the BRK (BREAK) instruction.

Lines 100 and 110 set up the RESLO and RESHI storage areas. Note that these areas are set up with the low byte first, followed by the high byte. This is the standard 6502 storage format for two-byte values, and it's a good idea to get accustomed to it.

Multi-byte subtraction also works the same way as the single-byte version, except that the first subtract operation is preceded by a SEC (SET CARRY) instruction. **Figure 9** shows an example of the three-byte subtract operation \$4203F5 - \$2E45FF. When finished, the result will be placed in RESL (LOW ORDER), RESM (MIDDLE) and RESH (HIGH ORDER). Try executing this code and observe that the resulting number is \$13BDF6.

```
01      *=  $600
10      CLD                ;BINARY MODE
20      LDA #$F5           ;GET OP1 LOW
30      SEC                ;FIRST SUBTRACT
40      SBC #$FF           ;SUB OP2 LOW
50      STA RESL           ;SAVE LOW RESULT
60      LDA #$03           ;GET OP1 MIDDLE
70      SBC #$45           ;SUB OP2 MIDDLE
80      STA RESM           ;SAVE MID RESULT
90      LDA #$42           ;GET OP1 HI
0100       SBC #$2E        ;SUB OP2 HIGH
0110       STA RESH        ;SAVE HIGH RESULT
0120       BRK             ;ALL DONE!
0130 RESL *=*+1            ;LOW RESULT BYTE
0140 RESM *=*+1            ;MID RESULT BYTE
0150 RESH *=*+1            ;HIGH RESULT BYTE
0160       .END            ;END OF ASSEMBLY
```

Figure 9.

What about the decimal mode?

Remember how the 6502 uses two different methods of storing numbers? We have been looking at multi-byte operations in the binary mode. Multi-byte decimal mode math works *exactly* like binary, but the data is stored in binary-coded decimal (see Issue 15 for a discussion of BCD). All you have to do to select BCD math is use the SED (SET DECIMAL MODE) instruction at the start of your program. You can return to binary math at any time by using the CLD (CLEAR DECIMAL MODE) instruction.

Now that we've looked at the basics of multi-byte math, let's make a few generalizations about the process.

```
10      LDA BYTE1A         ;BYTE 1
15      CLC                ;ON FIRST ONLY!
20      ADC BYTE1B
25      STA RESULT1
30      LDA BYTE2A         ;BYTE2
35      ADC BYTE2B
```

```
40    STA RESULT2
45     .
50     .             ;ETC.
55     .
60    LDA BYTEnA     ;BYTE n
65    ADC BYTEnB
70    STA RESULTn
```

Figure 10.

Figure 10 shows the procedure for a multi-byte add, where n is the number of bytes in the value. Note that the CLC instruction is used only for the *first* add of the group.

```
10    LDA BYTE1A     ;BYTE 1
15    SEC            ;ON FIRST ONLY!
20    SBC BYTE1B
25    STA RESULT1
30    LDA BYTE2A     ;BYTE2
35    SBC BYTE2B
40    STA RESULT2
45     .
50     .             ;ETC.
55     .
60    LDA BYTEnA     ;BYTE n
65    SBC BYTEnB
70    STA RESULTn
```

Figure 11.

Figure 11 shows the procedure for a multi-byte subtract, where n is the number of bytes in the value. The subtract procedure is similar to the add in that the SEC instruction is only used for the *first* subtract.

What happens when you want to add or subtract two values of different length, such as adding a one-byte value to a three-byte value? **Figure 12** shows how this is done.

```
10     *=  $600
15     CLD              ;BINARY MODE
20     LDA SCORE        ;GET SCORE LO
25     CLC              ;CLEAR 1ST TIME
30     ADC POINTS       ;ADD POINTS
35     STA SCORE        ;SAVE SCORE LO
40     LDA SCORE+1      ;GET SCORE MID
45     ADC #0           ;ADD DUMMY ZERO
50     STA SCORE+1      ;SAVE SCORE MID
55     LDA SCORE+2      ;GET SCORE HIGH
60     ADC #0           ;ADD DUMMY ZERO
65     STA SCORE+2      ;SAVE SCORE HIGH
70     BRK              ;ALL DONE!
75 POINTS *=*+1         ;ONE BYTE
80 SCORE  *=*+3         ;THREE BYTES
85     .END             ;END OF ASSEMBLY
```

Figure 12.

(Continued on next page.)

The program in **Figure 12** adds the one-byte value POINTS to the three-byte value SCORE. In this example the three bytes of SCORE are not individually labeled, but are referenced as SCORE (LOW ORDER), SCORE+1 (MIDDLE) and SCORE+2 (HIGH ORDER). The +1 and +2 added to the label SCORE simply indicate that the assembler is to add 1 and 2 to the address of SCORE for these operations. For example, if SCORE is located at $4000, SCORE+1 is address $4001, and SCORE+2 is $4002. If we had indicated SCORE-1, the address used would be $3FFF.

By looking at **Figure 12**, you will see that the first ADC operation adds the low byte of SCORE to POINTS, placing the result in SCORE. This is a typical first add, with a CLC operation before the addition.

The second and third adds are special in this case. Since POINTS is a one-byte field and SCORE is a three-byte field, we must complete the last two additions as if POINTS were three bytes long. As you can see from the example, the second and third adds simply add zeros to the second and third bytes of SCORE. This ensures that any carries out of the low bytes of SCORE will be properly taken care of. By adding zeros, the only factor affecting the result is the carry flag.

The challenge.

No tutorial would be complete without a challenge to the readers. For next month try to solve the following problems.

PROBLEM 1: Subtract the two-byte field WITHD (withdrawals) from the three-byte field OLDBAL (old balance), placing the result in the three-byte field NEWBAL (new balance). All fields should be stored in BCD, with standard data storage formats. Start with OLDBAL = 108673 and WITHD = 4285. After the subtraction is complete, check NEWBAL to be sure it contains 104388.

PROBLEM 2: Start with three 10-byte tables. Label these tables TABLE1, TABLE2 and TABLE3. Initialize TABLE1 and TABLE2 as follows:

```
TABLE1 .BYTE $10,$18,$40,$86,$9A
       .BYTE $A0,$BC,$C0,$F0,$F8
TABLE2 .BYTE $00,$08,$14,$2F,$9A
       .BYTE $90,$0B,$22,$65,$78
```

Write the code necessary to subtract each byte of TABLE2 from the corresponding byte of TABLE1, placing the result in TABLE3. That is, subtract the first byte of TABLE2 from the first byte of TABLE1 and place it in the first byte of TABLE3. Repeat this process for each of the ten bytes in the tables. When complete, TABLE3 should contain the values:

```
$10,$10,$2C,$57,$00,$10,$B1,$9E,$8B,$80
```

These problems should get you thinking about multi-byte operations more deeply. Whatever you do, *don't give up*! Stick with it and you'll soon get the hang of it.

Next month, we'll start looking at the many ways to manipulate our friend, the eight-bit byte. □

We just made owning an Atari computer a lot more logical.

Introducing the Rana 1000 disk drive. It's a whole new game for Atari computers.

This two digit LED readout displays a code that tells you everything you need to know.

This beeping button tells you your write protect feature is keeping your information safe.

The remaining buttons beep when touched, and provide readouts on density storage, error status, and drive number.

This button beeps when you touch it, and the LED readout tells you what track you're on.

When Rana Systems introduced the Elite Series of Apple® compatible disk drives, we didn't know what a tremendous impact they would make. It turned out to be a line so outstanding in performance, styling, capacity, and price, that it instantaneously made us a major force in the market. Well, needless to say, the response was so great that we were forced to create the same highly advanced disk drive for Atari.® A disk drive that when coupled with Atari's computer, could perform everything from accounting, financial planning, and stock charting, to word processing, business management, and letting you write your own programs. Plus, we made it simple enough for a child to use, for learning anything from the alphabet to a foreign language.

Working with a diskette versus playing with a cassette.

Let's face it. The only reason Atari made a cassette option to their computer was to make it affordable. But now you don't have to settle for less. Because now you can get a diskette for your Atari computer which outperforms their cassette and costs ⅓ less than their disk drive. With Atari's cassette you only get half the functions of a computer compared to what our floppy disk can give you. Their cassette is not only limited in the software available, but it also takes 20 times longer to get the information you need. And Rana's disk drive offers twice the storage capacity of either their cassette or disk drive.

Why even stylewise our new low profile design not only looks 100 times more spectacular, but it occupies 3 times less space. And our new Rana 1000 also gives you a piece of its mind every time you use it, because our disk drive gives you information as well as takes it. And we think that says a lot.

The disk drive that has all the answers.

Rana offers you a myriad of features Atari couldn't even conceive of. Like five electronic functions on the front panel that actually beep and give you a LED readout when touched. Our disk drive tells you what track you're on, and what density and how much information you're storing. It lets you switch from a single density of 90,000 letters to a double density of 180,000 letters, on a single diskette. And, we have a write protect feature which protects your diskette from being erased. In fact, no other disk drive can offer you that.

As you can see, it was easy to build a disk drive superior to Atari's. Because for every reason you buy a disk drive, Rana has superior technology.

The Rana 1000 disk drive. It brings your Atari computer to a higher level of sophistication for a price one third lower than Atari's. So your choice shouldn't even be a matter of logic.

Just common sense.

RanaSystems *Always a step ahead.*

21300 Superior Street, Chatsworth, CA 91311 213-709-5484. For dealer information call toll free: 1-800-421-2207. In California only call: 1-800-262-1221. Source Number: TCT-654

INDEX TO ADVERTISERS

This index is an additional service. While every effort is made to provide a complete and accurate listing, the publisher cannot be responsible for inadvertent errors.

NEXT ISSUE:

ADVENTURE!

- Munch'In Climb'In
- Crash Dive! (Game)
- Communications for the Handicapped
- Adventure Games Reviewed

Coming in the April ANALOG Computing.


John Anderson's
RALLY SPEEDWAY
SLIP INTO SOMETHING TURBO-CHARGED
like the road-scorching speedster in RALLY SPEEDWAY — the game that turns your ATARI into a full-blown, four-wheeled demon! Take on our demanding course, or use the unique Construct-A-Track option to design your own. Push yourself to the limit for a better lap time, or get a friend in on the action for a one-on-one duel to the finish line — it's your choice!
CALL TOLL FREE 1-800-327-7172
MAKE TRACKS TO YOUR DEALER TODAY!
ATARI 400/800/1200 CARTRIDGE
053-0171 . . . . $49.95
If your dealer doesn't have Rally Speedway in stock, phone 1-800-327-7172 with your order, or write us at Adventure International, P.O. Box 3435, Longwood, FL 32750 for your FREE catalog.
DEALER ORDERS WELCOME
155
COPYRIGHT 1983
Adventure International
A DIVISION OF SCOTT ADAMS, INC.